

B/10/9
5/9/13

Docket No.: 43876-128

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of

Craig C. HANSEN, et al.

Serial No.: 09/922,319

Group Art Unit: 2671

Filed: August 2, 2001

Examiner: M. Monestime

For: SYSTEM WITH WIDE OPERAND ARCHITECTURE AND METHOD

LETTER TO DRAFTSMAN

Hon. Assistant Commissioner for Patents
Washington, DC 20231

Sir:

In response to the Office action mailed on September 23, 2002, having a shortened statutory response period set to expire on December 23, 2002, a three (3) month extension of time up to and including March 23, 2003 being submitted herewith this Amendment, please amend the above identified application as follows:

IN THE DRAWINGS

Please add Figures 12A-34E in accordance with the attached sheets of Figures 12A-20E.

If there are any outstanding issues that might be resolved by an interview or an Examiner's amendment, the Examiner is requested to call Applicants' attorney at the telephone number shown below.

Date: March 24, 2003

Respectfully submitted,
McDERMOTT, WILL & EMERY
By: [Signature]
Lawrence T. Cullen
Registration No. 44,489

600 13th Street, N.W. , Suite 1200
Washington, D.C. 20006-3096
Telephone: (202) 756-8000
Facsimile: (202) 756-8087

1260

Operation codes

W.SWITCH.B	Wide switch big-endian
W.SWITCH.L	Wide switch little-endian

Selection

class	op	order
Wide switch	W.SWITCH	B L

Format

W.op.order ra=rc,rd,rb

ra=woporder(rc,rd,rb)

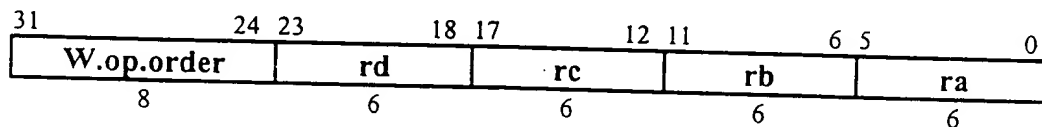
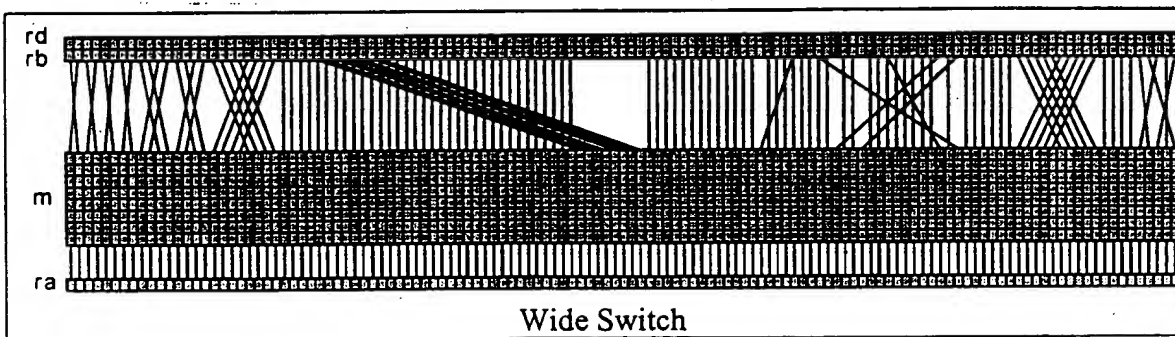


FIG. 12A

1230



F16. 12B

1250
↙

Definition

```

def WideSwitch(op,rd,rc,rb,ra)
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 64)
  b ← RegRead(rb, 128)
  if c1..0 ≠ 0 then
    raise AccessDisallowedByVirtualAddress
  elseif c6..0 ≠ 0 then
    VirtAddr ← c and (c-1)
    w ← wsize ← (c and (0-c)) || 01
  else
    VirtAddr ← c
    w ← wsize ← 128
  endif
  msize ← 8*wsize
  lsize ← log(wsize)
  case op of
    W.SWITCH.B:
      order ← B
    W.SWITCH.L:
      order ← L
  endcase
  m ← LoadMemory(c, VirtAddr, msize, order)
  db ← d || b
  for i ← 0 to 127
    j ← 0 || i lsize-1..0
    k ← m7*w+j || m6*w+j || m5*w+j || m4*w+j || m3*w+j || m2*w+j || mw+j || mj
    l ← i 7..lsize || j lsize-1..0
    ai ← dbl
  endfor
  RegWrite(ra, 128, a)
enddef

```

F16. 12C

1280
←

Exceptions

Access disallowed by virtual address

Access disallowed by tag

Access disallowed by global TB

Access disallowed by local TB

Access detail required by tag

Access detail required by local TB

Access detail required by global TB

Local TB miss

Global TB miss

F16.12D

1310

Operation codes

W.TRANSLATE.8.B	Wide translate bytes big-endian
W.TRANSLATE.16.B	Wide translate doublets big-endian
W.TRANSLATE.32.B	Wide translate quadlets big-endian
W.TRANSLATE.64.B	Wide translate octlets big-endian
W.TRANSLATE.8.L	Wide translate bytes little-endian
W.TRANSLATE.16.L	Wide translate doublets little-endian
W.TRANSLATE.32.L	Wide translate quadlets little-endian
W.TRANSLATE.64.L	Wide translate octlets little-endian

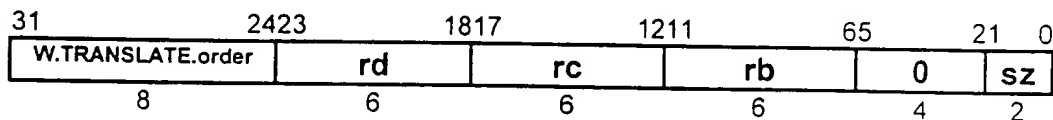
Selection

class	size	order
Wide translate	8 16 32 64	B L

Format

W.TRANSLATE.size.order rd=rc,rb

rd=wtranslatesizeorder(rc,rb)



sz ← log(size) - 3

FIG. 13A

133°

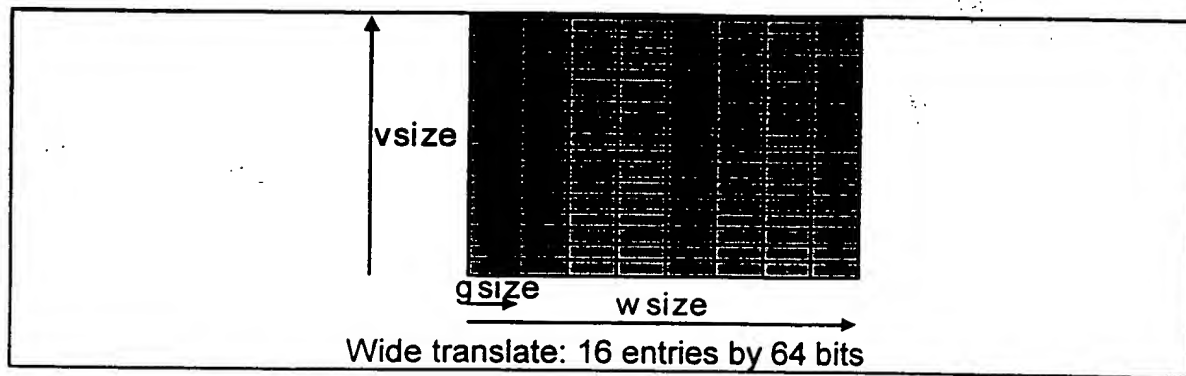


FIG. 13B

1350

Definition

```

def WideTranslate(op, gsize, rd, rc, rb)
  c ← RegRead(rc, 64)
  b ← RegRead(rb, 128)
  lsize ← log(gsize)
  if csize-4..0 ≠ 0 then
    raise AccessDisallowedByVirtualAddress
  endif
  if c4..lsize-3 ≠ 0 then
    wsize ← (c and (0-c)) || 03
    t ← c and (c-1)
  else
    wsize ← 128
    t ← c
  endif
  lwsiz ← log(wsize)
  if twsize+4..lwsiz-2 ≠ 0 then
    msiz ← (t and (0-t)) || 04
    VirtAddr ← t and (t-1)
  else
    msiz ← 256*wsize
    VirtAddr ← t
  endif
  case op of
    W.TRANSLATE.B:
      order ← B
    W.TRANSLATE.L:
      order ← L
  endcase
  m ← LoadMemory(c, VirtAddr, msiz, order)
  vsiz ← msiz/wsize
  lvsiz ← log(vsiz)
  for i ← 0 to 128-gsize by gsize
    j ← ((order=B)lvsiz^(blvsiz-1+i..i))*wsize+iwsize-1..0
    agsize-1+i..i ← mj+gsize-1..j
  endfor
  RegWrite(rd, 128, a)
enddef

```

F16. 130

← 1380

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

F16. 13D

W.MUL.MAT.8.B	Wide multiply matrix signed byte big-endian
W.MUL.MAT.8.L	Wide multiply matrix signed byte little-endian
W.MUL.MAT.16.B	Wide multiply matrix signed doublet big-endian
W.MUL.MAT.16.L	Wide multiply matrix signed doublet little-endian
W.MUL.MAT.32.B	Wide multiply matrix signed quadlet big-endian
W.MUL.MAT.32.L	Wide multiply matrix signed quadlet little-endian
W.MUL.MAT.C.8.B	Wide multiply matrix signed complex byte big-endian
W.MUL.MAT.C.8.L	Wide multiply matrix signed complex byte little-endian
W.MUL.MAT.C.16.B	Wide multiply matrix signed complex doublet big-endian
W.MUL.MAT.C.16.L	Wide multiply matrix signed complex doublet little-endian
W.MUL.MAT.M.8.B	Wide multiply matrix mixed-signed byte big-endian
W.MUL.MAT.M.8.L	Wide multiply matrix mixed-signed byte little-endian
W.MUL.MAT.M.16.B	Wide multiply matrix mixed-signed doublet big-endian
W.MUL.MAT.M.16.L	Wide multiply matrix mixed-signed doublet little-endian
W.MUL.MAT.M.32.B	Wide multiply matrix mixed-signed quadlet big-endian
W.MUL.MAT.M.32.L	Wide multiply matrix mixed-signed quadlet little-endian
W.MUL.MAT.P.8.B	Wide multiply matrix polynomial byte big-endian
W.MUL.MAT.P.8.L	Wide multiply matrix polynomial byte little-endian
W.MUL.MAT.P.16.B	Wide multiply matrix polynomial doublet big-endian
W.MUL.MAT.P.16.L	Wide multiply matrix polynomial doublet little-endian
W.MUL.MAT.P.32.B	Wide multiply matrix polynomial quadlet big-endian
W.MUL.MAT.P.32.L	Wide multiply matrix polynomial quadlet little-endian
W.MUL.MAT.U.8.B	Wide multiply matrix unsigned byte big-endian
W.MUL.MAT.U.8.L	Wide multiply matrix unsigned byte little-endian
W.MUL.MAT.U.16.B	Wide multiply matrix unsigned doublet big-endian
W.MUL.MAT.U.16.L	Wide multiply matrix unsigned doublet little-endian
W.MUL.MAT.U.32.B	Wide multiply matrix unsigned quadlet big-endian
W.MUL.MAT.U.32.L	Wide multiply matrix unsigned quadlet little-endian

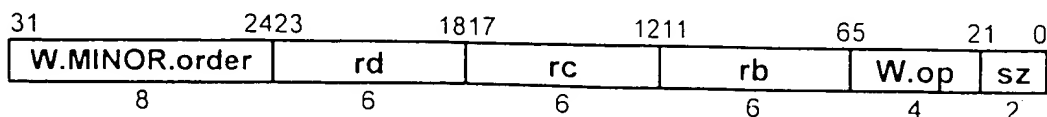
Selection

class	op	type	size	order
multiply	W.MUL.MAT	NONE M U P	8 16 32	B
				L
		C	8 16	B
				L

Format

W.op.size.order rd=rc,rb

rd=wopsizorder(rc,rb)



sz ← log(size) - 3

1430

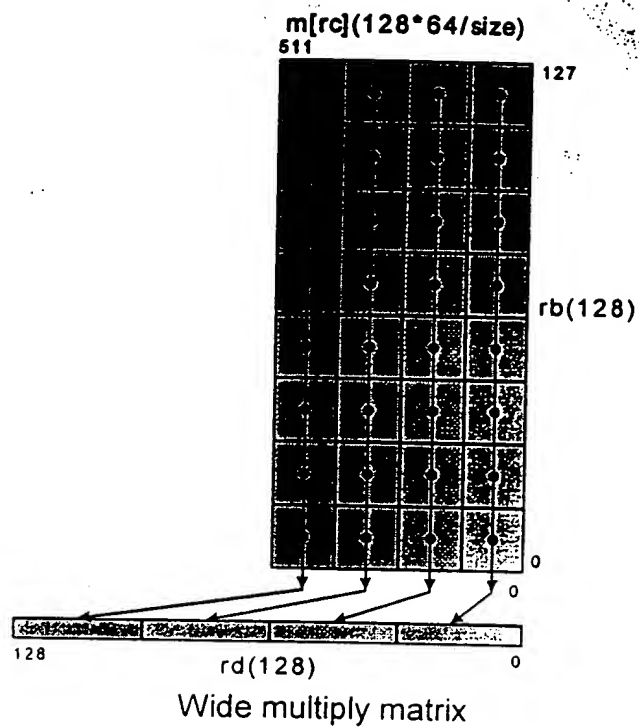


FIG. 14B

1460

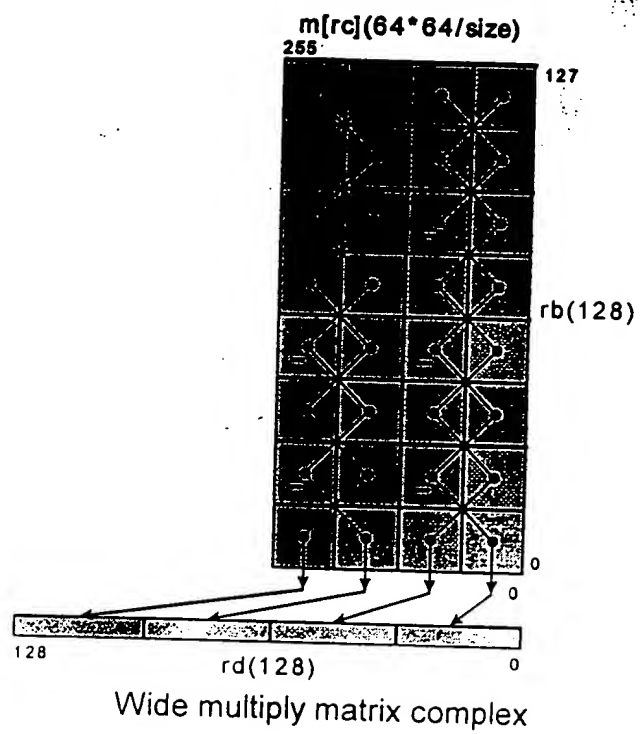


FIG. 14C

Definition

1480

```
def mul(size,h,vs,v,i,ws,w,j) as
  mul ← ((vs&vsize-1+i)h-size || vsize-1+i..i) * ((ws&wsize-1+j)h-size || wsize-1+j..j)
enddef

def c ← PolyMultiply(size,a,b) as
  p[0] ← 02*size
  for k ← 0 to size-1
    p[k+1] ← p[k] ^ ak ? (0size-k || b || 0k) : 02*size
  endfor
  c ← p[size]
enddef

def WideMultiplyMatrix(major,op,gsize,rd,rc,rb)
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 64)
  b ← RegRead(rb, 128)
  lsize ← log(gsize)
  if C1gsize-4..0 ≠ 0 then
    raise AccessDisallowedByVirtualAddress
  endif
  if C2..lsize-3 ≠ 0 then
    wsize ← (c and (0-c)) || 04
    t ← c and (c-1)
  else
    wsize ← 64
    t ← a
  endif
  lsize ← log(wsize)
  if tlsize+6-lsize..lsize-3 ≠ 0 then
    msize ← (t and (0-t)) || 04
    VirtAddr ← t and (t-1)
  else
    msize ← 128*wsize/gsize
    VirtAddr ← t
  endif
  case major of
    W.MINOR.B:
      order ← B
    W.MINOR.L:
      order ← L
  endcase
  case op of
    W.MUL.MAT.U.8, W.MUL.MAT.U.16, W.MUL.MAT.U.32, W.MUL.MAT.U.64:
      ms ← bs ← 0
    W.MUL.MAT.M.8, W.MUL.MAT.M.16, W.MUL.MAT.M.32, W.MUL.MAT.M.64:
      ms ← 0
      bs ← 1
    W.MUL.MAT.8, W.MUL.MAT.16, W.MUL.MAT.32, W.MUL.MAT.64,
    W.MUL.MAT.C.8, W.MUL.MAT.C.16, W.MUL.MAT.C.32, W.MUL.MAT.C.64:
      ms ← bs ← 1
    W.MUL.MAT.P.8, W.MUL.MAT.P.16, W.MUL.MAT.P.32, W.MUL.MAT.P.64:
  endcase
```

1480

```

m ← LoadMemory(c,VirtAddr,msize,order)
h ← 2*gszsize

for i ← 0 to wsize-gsize by gsize
  q[0] ← 02*gszsize
  for j ← 0 to vsize-gsize by gsize
    case op of
      W.MUL.MAT.P.8, W.MUL.MAT.P.16, W.MUL.MAT.P.32, W.MUL.MAT.P.64:
        k ← i+wszsize*j8..lgszsize
        q[j+gszsize] ← q[j] ^ PolyMultiply(gszsize,mk+gszsize-1..k,bj+gszsize-1..j)
      W.MUL.MAT.C.8, W.MUL.MAT.C.16, W.MUL.MAT.C.32, W.MUL.MAT.C.64:
        if (~i) & j & gszsize = 0 then
          k ← i-(j&gszsize)+wszsize*j8..lgszsize+1
          q[j+gszsize] ← q[j] + mul(gszsize,h,ms,m,k,bs,b,j)
        else
          k ← i+gszsize+wszsize*j8..lgszsize+1
          q[j+gszsize] ← q[j] - mul(gszsize,h,ms,m,k,bs,b,j)
        endif
      W.MUL.MAT.8, W.MUL.MAT.16, W.MUL.MAT.32, W.MUL.MAT.64,
      W.MUL.MAT.M.8, W.MUL.MAT.M.16, W.MUL.MAT.M.32,
W.MUL.MAT.M.64,
      W.MUL.MAT.U.8, W.MUL.MAT.U.16, W.MUL.MAT.U.32, W.MUL.MAT.U.64:
        q[j+gszsize] ← q[j] + mul(gszsize,h,ms,m,i+wszsize*j8..lgszsize,bs,b,j)
    endfor
    a2*gszsize-1+2*i..2*i ← q[vsize]
  endfor
  a127..2*wszsize ← 0
  RegWrite(rd, 128, a)
enddef

```

F16 14D (CONTINUED)

1490

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 14E

1510

Operation codes

W.MUL.MAT.X.B	Wide multiply matrix extract big-endian
W.MUL.MAT.X.L	Wide multiply matrix extract little-endian

Selection

class	op	order
Multiply matrix extract	W.MUL.MAT.X	B L

Format

W.op.order ra=rc,rd,rb

ra=wop(rc,rd,rb)

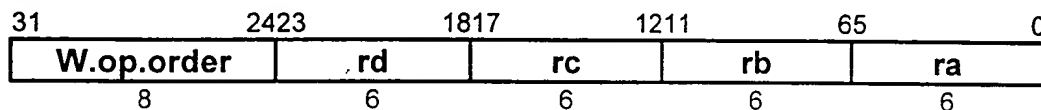


FIG. 15A

1520

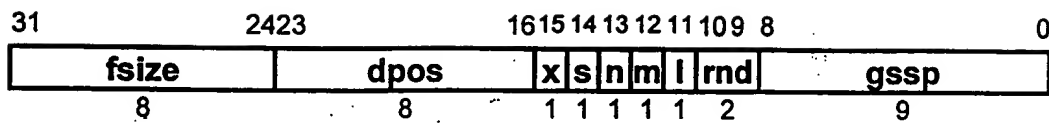
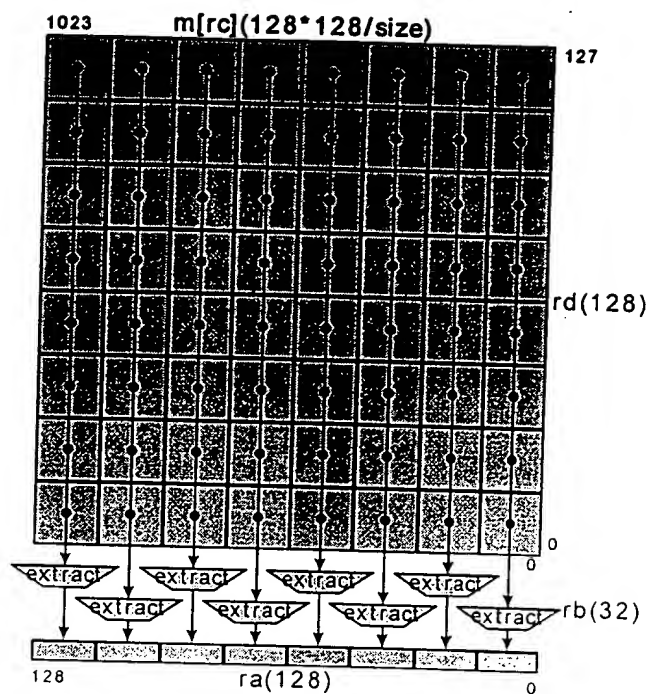


FIG. 15B

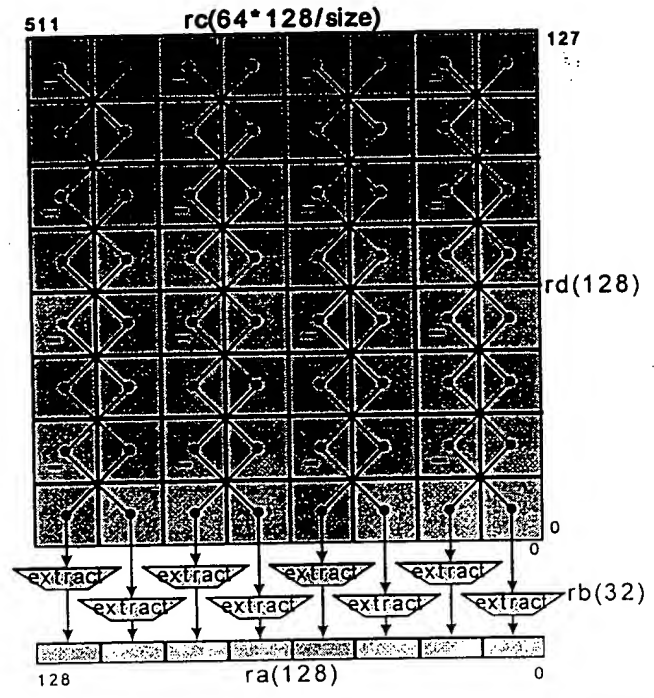
1530



Wide multiply matrix extract doublets

FIG. 15C

1560



Wide multiply matrix extract complex doublets

FIG. 15D

Definition

def mul(size,h,vs,v,i,ws,w,j) as
mul ← ((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) * ((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
enddef

def WideMultiplyMatrixExtract(op,ra,rb,rc,rd)

d ← RegRead(rd, 128)
c ← RegRead(rc, 64)
b ← RegRead(rb, 128)
case b_{8..0} of
0..255:
sgsize ← 128
256..383:
sgsize ← 64
384..447:
sgsize ← 32
448..479:
sgsize ← 16
480..495:
sgsize ← 8
496..503:
sgsize ← 4
504..507:
sgsize ← 2
508..511:
sgsize ← 1

endcase

l ← b₁₁

m ← b₁₂

n ← b₁₃

signed ← b₁₄

if c_{3..0} ≠ 0 then

wsiz ← (c and (0-c)) || 0⁴

t ← c and (c-1)

else

wsiz ← 128

t ← c

endif

if sgsiz < 8 then

gsiz ← 8

elseif sgsiz > wsiz/2 then

gsiz ← wsiz/2

else

gsiz ← sgsiz

endif

lgiz ← log(gsiz)

lwiz ← log(wsiz)

if t_{lwiz+6-n-lgiz..lwiz-3} ≠ 0 then

msiz ← (t and (0-t)) || 0⁴

VirtAddr ← t and (t-1)

else

msiz ← 64*(2-n)*wsiz/gsiz

VirtAddr ← t

endif

1580

```

vsize ← (1+n)*msize*gsize/wsize
mm ← LoadMemory(c,VirtAddr,msize,order)
h ← (2*gsize) + 7 - lsize
lmsize ← log(msize)
if (VirtAddrmsize-4..0 ≠ 0 then
    raise AccessDisallowedByVirtualAddress
endif
case op of
    W.MUL.MAT.X.B:
        order ← B
    W.MUL.MAT.X.L:
        order ← L
endcase
ms ← signed
ds ← signed ^ m
as ← signed or m
spos ← (b8..0) and (2*gsize-1)
dpos ← (0 || b23..16) and (gsize-1)
r ← spos
sfsz ← (0 || b31..24) and (gsize-1)
tfsz ← (sfsz = 0) or ((sfsz+dpos) > gsize) ? gsize-dpos : sfsz
fsz ← (tfsz + spos > h) ? h - spos : tfsz
if (b10..9 = Z) & ~signed then
    rnd ← F
else
    rnd ← b10..9
endif

```

FIG. 15E (CONTINUED)

1580

```

for i ← 0 to wsize-gsize by gsize
  q[0] ← 02*gsize+7-lgsize
  for j ← 0 to vsize-gsize by gsize
    if n then
      if (~i) & j & gsize = 0 then
        k ← i-(j&gsize)+wsize*j8..lgsize+1
        q[j+gsize] ← q[j] + mul(gsize,h,ms,mm,k,ds,d,j)
      else
        k ← i+gsize+wsize*j8..lgsize+1
        q[j+gsize] ← q[j] - mul(gsize,h,ms,mm,k,ds,d,j)
      endif
    else
      q[j+gsize] ← q[j] + mul(gsize,h,ms,mm,i+j*wsize/gsize,ds,d,j)
    endif
  endfor
  p ← q[128]
  case rnd of
    none, N:
      s ← 0h-r || ~pr || prr-1
    Z:
      s ← 0h-r || ph-1r
    F:
      s ← 0h
    C:
      s ← 0h-r || 1r
  endcase
  v ← ((ds & ph-1) || p) + (0 || s)

  if (vh..r+fsize = (as & vr+fsize-1)h+1-r-fsize) or not I then
    w ← (as & vr+fsize-1)gsize-fsize-dpos || vfsize-1+r..r || 0dpos
  else
    w ← (s ? (vh || ~vhgsize-dpos-1) : 1gsize-dpos) || 0dpos
  endif
  asize-1+i..i ← w
endfor
a127..wsize ← 0
RegWrite(ra, 128, a)
enddef

```

File (CONTINUED)

1580
↙

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

F16.15 F

1610

Operation codes

W.MUL.MAT.X.I.8.B	Wide multiply matrix extract immediate signed bytes big-endian
W.MUL.MAT.X.I.8.L	Wide multiply matrix extract immediate signed bytes little-endian
W.MUL.MAT.X.I.16.B	Wide multiply matrix extract immediate signed doublets big-endian
W.MUL.MAT.X.I.16.L	Wide multiply matrix extract immediate signed doublets little-endian
W.MUL.MAT.X.I.32.B	Wide multiply matrix extract immediate signed quadlets big-endian
W.MUL.MAT.X.I.32.L	Wide multiply matrix extract immediate signed quadlets little-endian
W.MUL.MAT.X.I.64.B	Wide multiply matrix extract immediate signed octlets big-endian
W.MUL.MAT.X.I.64.L	Wide multiply matrix extract immediate signed octlets little-endian
W.MUL.MAT.X.I.C.8.B	Wide multiply matrix extract immediate complex bytes big-endian
W.MUL.MAT.X.I.C.8.L	Wide multiply matrix extract immediate complex bytes little-endian
W.MUL.MAT.X.I.C.16.B	Wide multiply matrix extract immediate complex doublets big-endian
W.MUL.MAT.X.I.C.16.L	Wide multiply matrix extract immediate complex doublets little-endian
W.MUL.MAT.X.I.C.32.B	Wide multiply matrix extract immediate complex quadlets big-endian
W.MUL.MAT.X.I.C.32.L	Wide multiply matrix extract immediate complex quadlets little-endian

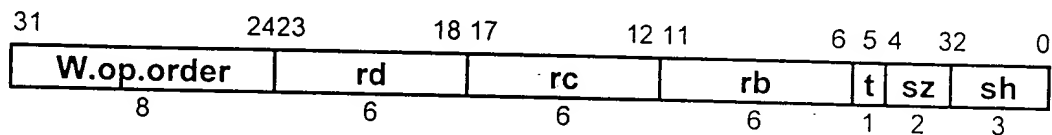
Selection

class	op	type	size	order
wide multiply matrix extract immediate	W.MUL.MAT.X.I	NONE	8 16 32 64	L B
		C	8 16 32	L B

Format

W.op.tsize.order rd=rc,rb,i

rd=wopsizeorder(rc,rb,i)



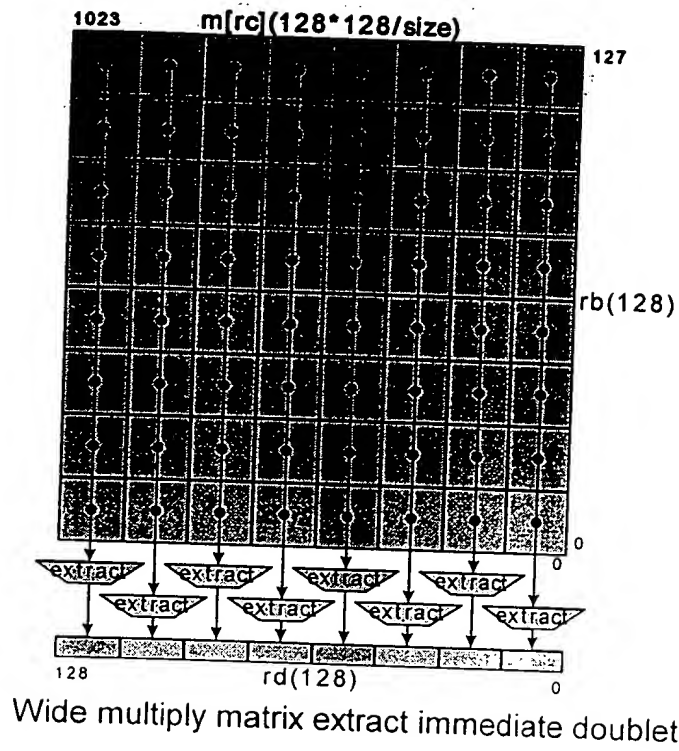
sz ← log(size) - 3

assert size+3 ≥ i ≥ size-4

sh ← i - size

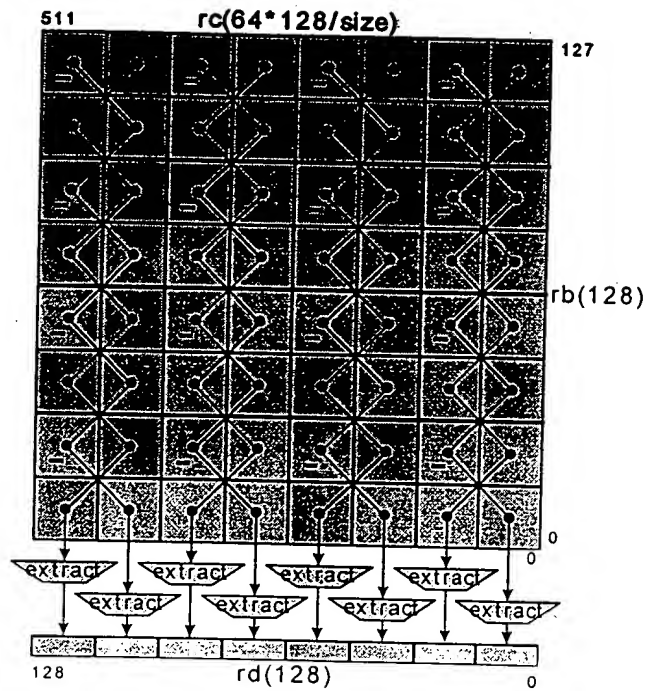
F16.16A

1630
1630



F16.16B

1660



Wide multiply matrix extract immediate complex doublets

F16 16C

Definition

def mul(size,h,vs,v,i,ws,w,j) as
mul $\leftarrow ((vs \& v_{size-1+i})^{h-size} \parallel v_{size-1+i..i}) * ((ws \& w_{size-1+j})^{h-size} \parallel w_{size-1+j..j})$
enddef

def WideMultiplyMatrixExtractImmediate(op,type,gsize,rd,rc,rb,sh)

c \leftarrow RegRead(rc, 64)

b \leftarrow RegRead(rb, 128)

lgsize \leftarrow log(gsize)

case type of

NONE:

if $c_{lgsize-4..0} \neq 0$ then

raise AccessDisallowedByVirtualAddress

endif

if $c_{3..lgsize-3} \neq 0$ then

wsiz $\leftarrow (c \text{ and } (0-c)) \parallel 0^4$

t $\leftarrow c \text{ and } (c-1)$

else

wsiz \leftarrow 128

t \leftarrow c

endif

lwsiz \leftarrow log(wsiz)

if $t_{lwsiz+6-lgsize..lwsiz-3} \neq 0$ then

msiz $\leftarrow (t \text{ and } (0-t)) \parallel 0^4$

VirtAddr $\leftarrow t \text{ and } (t-1)$

else

msiz $\leftarrow 128 * \text{wsiz} / \text{gsize}$

VirtAddr \leftarrow t

endif

vsize \leftarrow msiz * gsize / wsiz

C:

if $c_{lgsize-4..0} \neq 0$ then

raise AccessDisallowedByVirtualAddress

endif

if $c_{3..lgsize-3} \neq 0$ then

wsiz $\leftarrow (c \text{ and } (0-c)) \parallel 0^4$

t $\leftarrow c \text{ and } (c-1)$

else

wsiz \leftarrow 128

t \leftarrow c

endif

lwsiz \leftarrow log(wsiz)

if $t_{lwsiz+5-lgsize..lwsiz-3} \neq 0$ then

msiz $\leftarrow (t \text{ and } (0-t)) \parallel 0^4$

VirtAddr $\leftarrow t \text{ and } (t-1)$

else

msiz $\leftarrow 64 * \text{wsiz} / \text{gsize}$

VirtAddr \leftarrow t

endif

vsize $\leftarrow 2 * \text{msiz} * \text{gsize} / \text{wsiz}$

endcase

1680

```

case op of
  W.MUL.MAT.X.I.B:
    order ← B
  W.MUL.MAT.X.I.L:
    order ← L
endcase
as ← ms ← bs ← 1
m ← LoadMemory(c,VirtAddr,msize,order)
h ← (2*gsz) + 7 - lgsz - (ms and bs)
r ← gsz + (sh2 || sh)
for i ← 0 to wsize-gsz by gsz
  q[0] ← 02*gsz+7-lgsz
  for j ← 0 to vsize-gsz by gsz
    case type of
      NONE:
        q[j+gsz] ← q[j] + mul(gsz,h,ms,m,i+wsize*j8..lgsz,bs,b,j)
      C:
        if (~i) & j & gsz = 0 then
          k ← i-(j&gsz)+wsize*j8..lgsz+1
          q[j+gsz] ← q[j] + mul(gsz,h,ms,m,k,bs,b,j)
        else
          k ← i+gsz+wsize*j8..lgsz+1
          q[j+gsz] ← q[j] - mul(gsz,h,ms,m,k,bs,b,j)
        endif
      endcase
    endfor
    p ← q[vsize]
    s ← 0h-r || ~pr || prr-1
    v ← ((as & ph-1) || p) + (0 || s)
    if (vh..r+gsz = (as & vr+gsz-1)h+1-r-gsz then
      agsz-1+i..i ← vgsz-1+r..r
    else
      agsz-1+i..i ← as ? (vh || ~vhgsz-1) : 1gsz
    endif
  endfor
  a127..wsize ← 0
  RegWrite(rd, 128, a)
enddef

```

FIG. 16D (CONTINUED)

1690

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

F16 16 E

1710

Operation codes

W.MUL.MAT.C.F.16.B	Wide multiply matrix complex floating-point half big-endian
W.MUL.MAT.C.F.16.L	Wide multiply matrix complex floating-point half little-endian
W.MUL.MAT.C.F.32.B	Wide multiply matrix complex floating-point single big-endian
W.MUL.MAT.C.F.32.L	Wide multiply matrix complex floating-point single little-endian
W.MUL.MAT.F.16.B	Wide multiply matrix floating-point half big-endian
W.MUL.MAT.F.16.L	Wide multiply matrix floating-point half little-endian
W.MUL.MAT.F.32.B	Wide multiply matrix floating-point single big-endian
W.MUL.MAT.F.32.L	Wide multiply matrix floating-point single little-endian
W.MUL.MAT.F.64.B	Wide multiply matrix floating-point double big-endian
W.MUL.MAT.F.64.L	Wide multiply matrix floating-point double little-endian

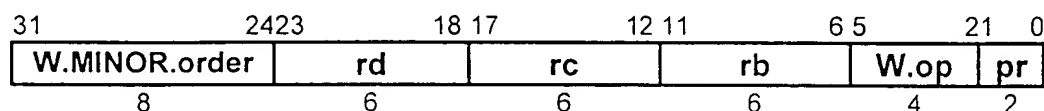
Selection

class	op	type	prec	order
wide multiply matrix	W.MUL.MAT	F	16 32 64	L B
		C.F	16 32	L B

Format

W.op.prec.order rd=rc,rb

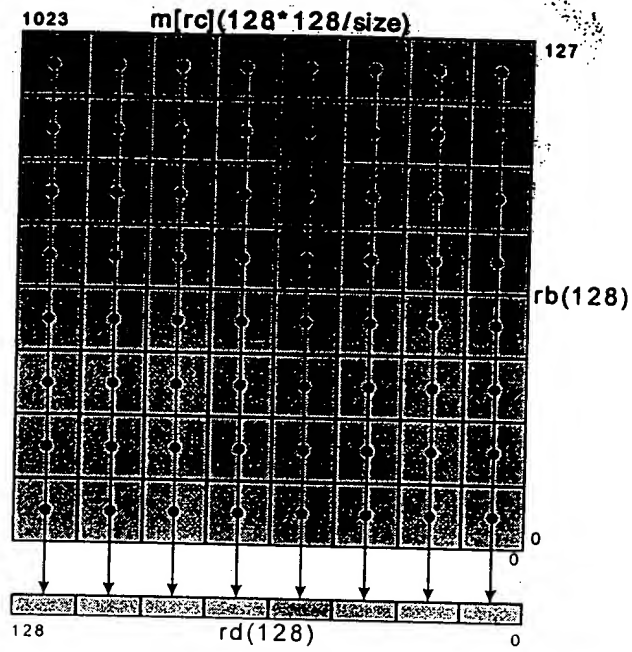
rd=wopprecorder(rc,rb)



pr ← log(prec) - 3

F16.17A

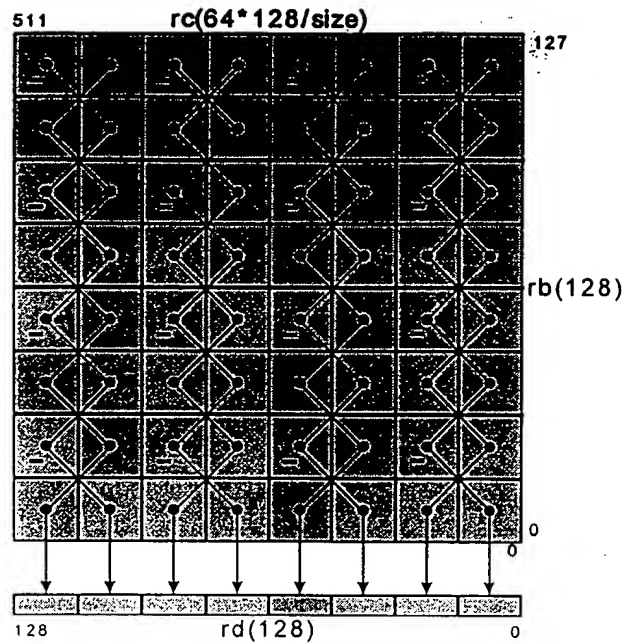
1730



Wide multiply matrix floating-point half

F16 .173

1760



Wide multiply matrix complex floating-point half

FIG. 17C

Definition

178

```
def mul(size,v,i,w,j) as
    mul ← fmul(F(size,vsize-1+i..i),F(size,wsize-1+j..j))
enddef

def WideMultiplyMatrixFloatingPoint(major,op,gsize,rd,rc,rb)
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 128)
    lgsiz ← log(gsize)
    switch op of
        W.MUL.MAT.F.16, W.MUL.MAT.F.32, W.MUL.MAT.F.64:
            if clgsiz-4..0 ≠ 0 then
                raise AccessDisallowedByVirtualAddress
            endif
            if c3..lgsiz-3 ≠ 0 then
                wsize ← (c and (0-c)) || 04
                t ← c and (c-1)
            else
                wsize ← 128
                t ← c
            endif
            lwsiz ← log(wsize)
            if tlwsiz+6-lgsiz..lwsiz-3 ≠ 0 then
                msiz ← (t and (0-t)) || 04
                VirtAddr ← t and (t-1)
            else
                msiz ← 128*wsize/gsize
                VirtAddr ← t
            endif
            vsiz ← msiz*gsize/wsize
        W.MUL.MAT.C.F.16, W.MUL.MAT.C.F.32, W.MUL.MAT.C.F.64:
            if clgsiz-4..0 ≠ 0 then
                raise AccessDisallowedByVirtualAddress
            endif
            if c3..lgsiz-3 ≠ 0 then
                wsize ← (c and (0-c)) || 04
                t ← c and (c-1)
            else
                wsize ← 128
                t ← c
            endif
            lwsiz ← log(wsize)
            if tlwsiz+5-lgsiz..lwsiz-3 ≠ 0 then
                msiz ← (t and (0-t)) || 04
                VirtAddr ← t and (t-1)
            else
                msiz ← 64*wsize/gsize
                VirtAddr ← t
            endif
            vsiz ← 2*msiz*gsize/wsize
    endcase
```

1780

```

case major of
  M.MINOR.B:
    order ← B
  M.MINOR.L:
    order ← L
endcase
m ← LoadMemory(c,VirtAddr,msize,order)
for i ← 0 to wsize-gsize by gsize
  q[0].t ← NULL
  for j ← 0 to vsize-gsize by gsize
    case op of
      W.MUL.MAT.F.16, W.MUL.MAT.F.32, W.MUL.MAT.F.64:
        q[j+gsize] ← fadd(q[j], mul(gsize,m,i+wsiz*j8..lgsiz,b,j))
      W.MUL.MAT.C.F.16, W.MUL.MAT.C.F.32, M.MUL.MAT.C.F.64:
        if (~i) & j & gsize = 0 then
          k ← i-(j&gsize)+wsiz*j8..lgsiz+1
          q[j+gsize] ← fadd(q[j], mul(gsize,m,k,b,j))
        else
          k ← i+gsize+wsiz*j8..lgsiz+1
          q[j+gsize] ← fsub(q[j], mul(gsize,m,k,b,j))
        endif
      endif
    endcase
  endfor
  a[gsiz-1+i..i] ← q[vsiz]
endfor
a[127..wsiz] ← 0
RegWrite(rd, 128, a)
enddef

```

FIG. 17D (CONTINUED)

1790

Exceptions

Floating-point arithmetic
Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG 17E

1810

Operation codes

W.MUL.MAT.G.8.B	Wide multiply matrix Galois bytes big-endian
W.MUL.MAT.G.8.L	Wide multiply matrix Galois bytes little-endian

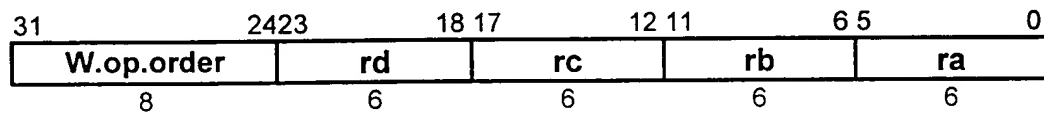
Selection

class	op	size	order
Multiply matrix Galois	W.MUL.MAT.G	8	B L

Format

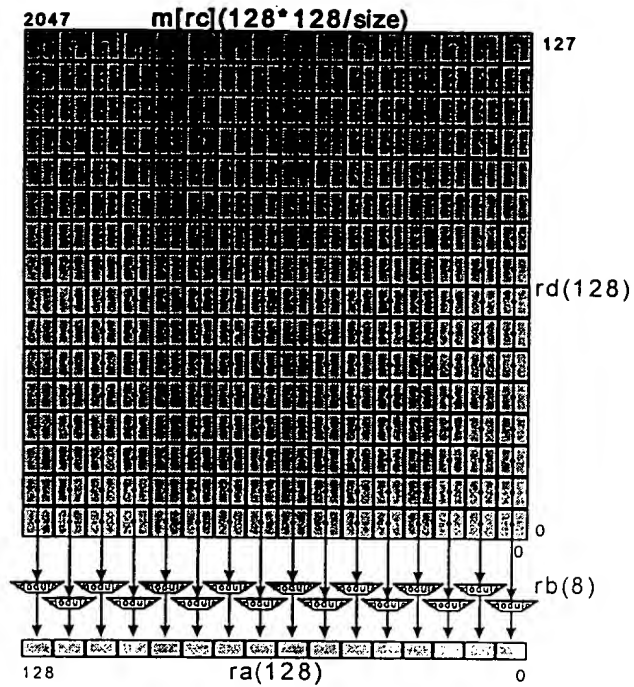
W.op.order ra=rc,rd,rb

ra=woporder(rc,rd,rb)



F16.18A

1830



Wide multiply matrix Galois byte

F16-18B

Definition

```

def c ← PolyMultiply(size,a,b) as
  p[0] ← 02*size
  for k ← 0 to size-1
    p[k+1] ← p[k] ^ ak ? (0size-k || b || 0k) : 02*size
  endfor
  c ← p[size]
enddef

def c ← PolyResidue(size,a,b) as
  p[0] ← a
  for k ← size-1 to 0 by -1
    p[k+1] ← p[k] ^ p[0]size+k ? (0size-k || 11 || b || 0k) : 02*size
  endfor
  c ← p[size]size-1..0
enddef

def WideMultiplyMatrixGalois(op,gsize,rd,rc,rb,ra)
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 64)
  b ← RegRead(rb, 128)
  lgsiz ← log(gsize)
  if clgsiz-4..0 ≠ 0 then
    raise AccessDisallowedByVirtualAddress
  endif
  if c3..lgsiz-3 ≠ 0 then
    wsize ← (c and (0-c)) || 04
    t ← c and (c-1)
  else
    wsize ← 128
    t ← c
  endif
  lwsiz ← log(wsize)
  if tlwsiz+6-lgsiz..lwsiz-3 ≠ 0 then
    msize ← (t and (0-t)) || 04
    VirtAddr ← t and (t-1)
  else
    msize ← 128*wsize/gsize
    VirtAddr ← t
  endif
  case op of
    W.MUL.MAT.G.8.B:
      order ← B
    W.MUL.MAT.G.8.L:
      order ← L
  endcase
enddef

```

1860

```

m ← LoadMemory(c, VirtAddr, msize, order)
for i ← 0 to wsize-gsize by gsize
  q[0] ← 02*gsize
  for j ← 0 to vsize-gsize by gsize
    k ← i+wsize*j*8..lgsize
    q[j+gsize] ← q[j] ^ PolyMultiply(gsize, mk+gsize-1..k, qj+gsize-1..j)
  endfor

  agsize-1+i..i ← PolyResidue(gsize, q[vsize], bgsize-1..0)
endfor
a127..wsize ← 0
RegWrite(ra, 128, a)
enddef

```

FIG. 18C (CONTINUED)

← 1890

Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 18D

1910

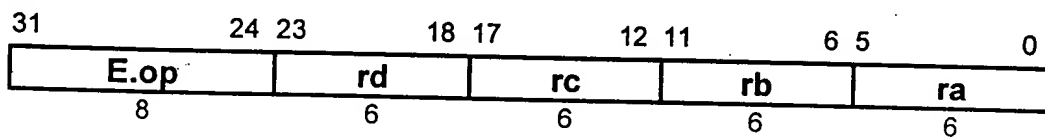
Operation codes

E.MUL.ADD.X	Ensemble multiply add extract
E.CON.X	Ensemble convolve extract

Format

E.op rd@rc,rb,ra

rd=gop(rd,rc,rb,ra)



F16.17A

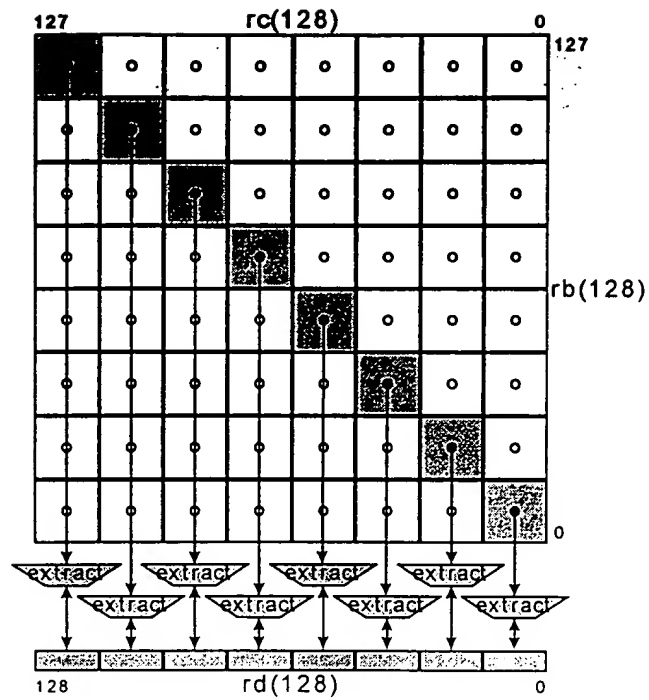
1910

Figures 19B and 20B has blank fields: should be .

fsiz	dpos	x	s	n	m	l	rnd	gssp
------	------	---	---	---	---	---	-----	------

FIG. 196

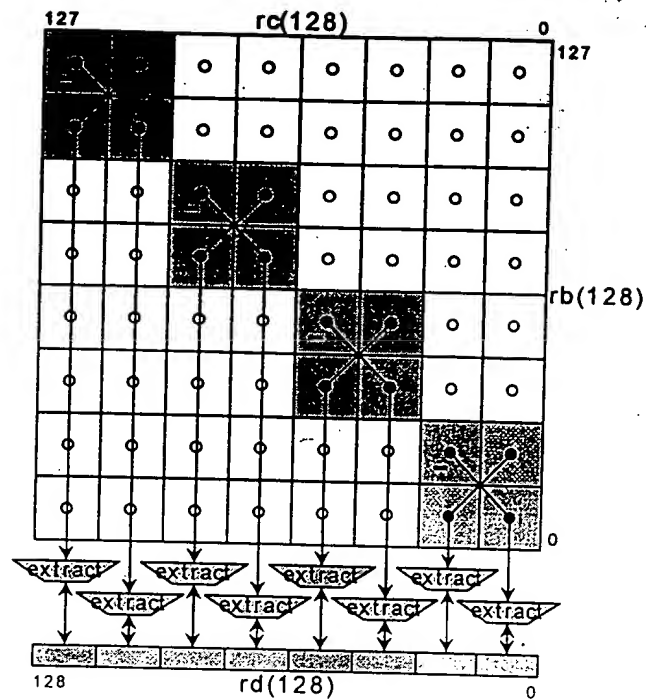
1930



Ensemble multiply add extract doublets

FIG. 19C

1945

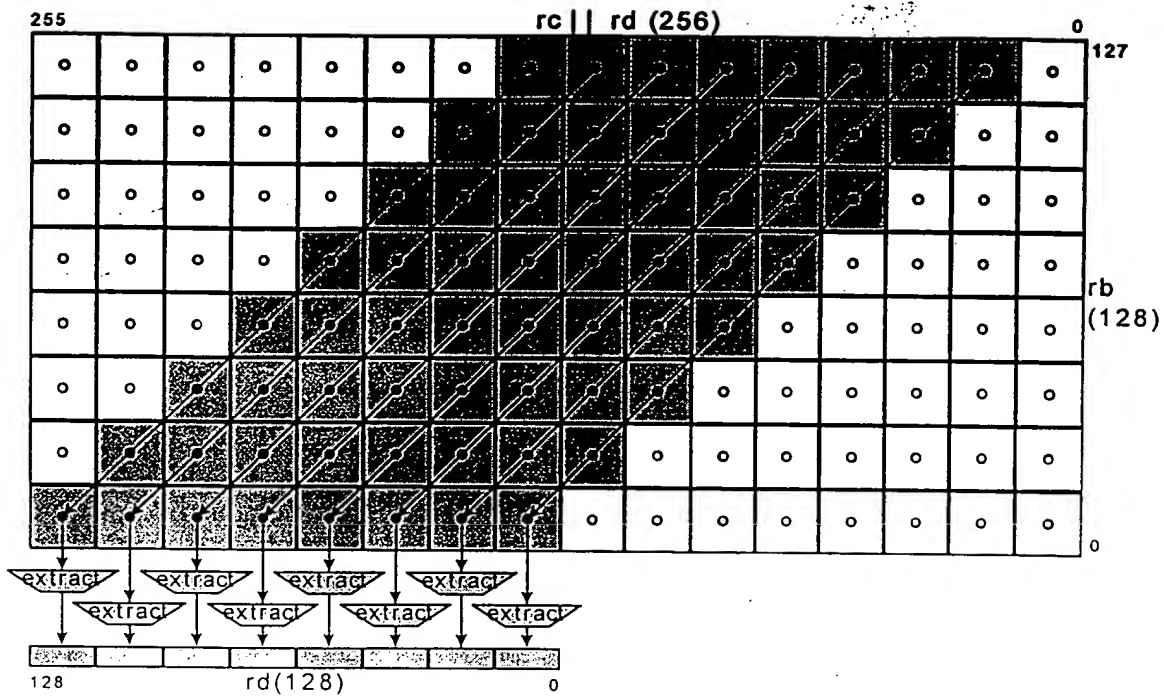


Ensemble complex multiply add extract doublets

The ensemble-multiply-add-extract instructions (E.MUL.ADD.X), when the x bit is set, multiply the low-order 64 bits of each of the rc and rb registers and produce extended (double-size) results.

F16.19D

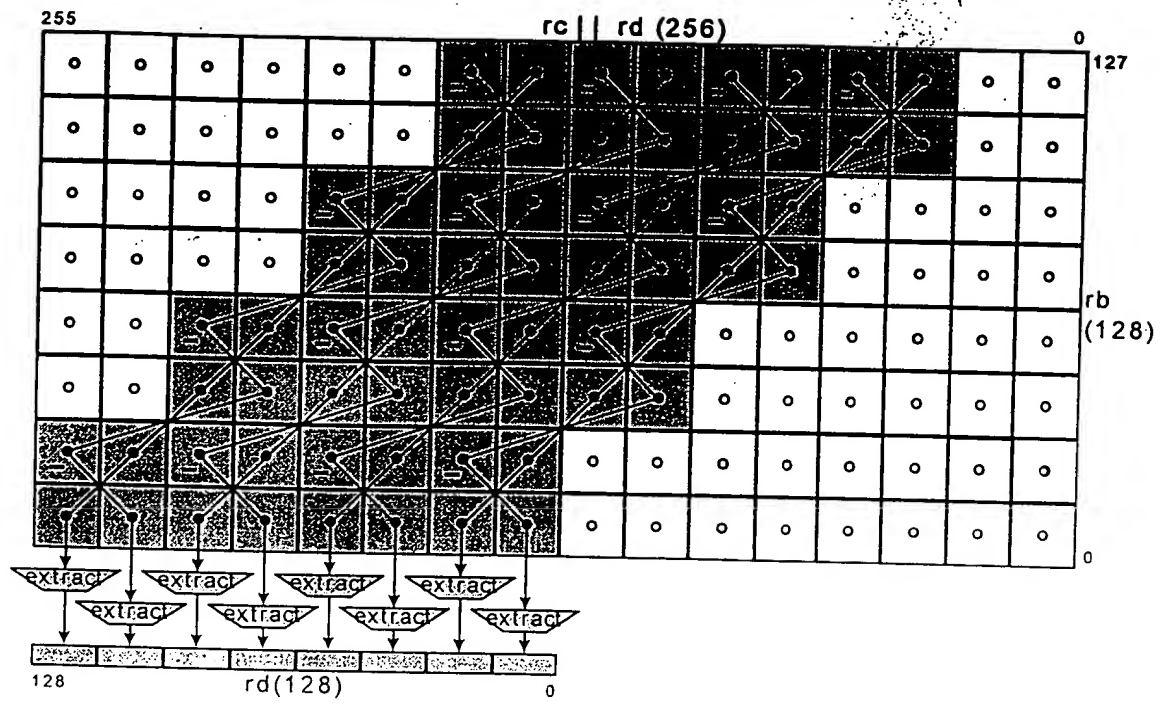
1960



Ensemble convolve extract doublets

FIG. 19E

1975



Ensemble convolve extract complex doublets

FIG. 19F

Definition

def mul(size,h,vs,v,i,ws,w,j) as
 mul $\leftarrow ((vs \& v_{size-1+i})^{h-size} \parallel v_{size-1+i..i}) * ((ws \& w_{size-1+j})^{h-size} \parallel w_{size-1+j..j})$
enddef

def EnsembleExtractInplace(op,ra,rb,rc,rd) as
 d \leftarrow RegRead(rd, 128)
 c \leftarrow RegRead(rc, 128)
 b \leftarrow RegRead(rb, 128)
 case a8..0 of
 0..255:
 sgsize \leftarrow 128
 256..383:
 sgsize \leftarrow 64
 384..447:
 sgsize \leftarrow 32
 448..479:
 sgsize \leftarrow 16
 480..495:
 sgsize \leftarrow 8
 496..503:
 sgsize \leftarrow 4
 504..507:
 sgsize \leftarrow 2
 508..511:
 sgsize \leftarrow 1
 endcase
 l \leftarrow a11
 m \leftarrow a12
 n \leftarrow a13
 signed \leftarrow a14
 x \leftarrow a15
 case op of
 E.CON.X:
 if (sgsize < 8) then
 gsize \leftarrow 8
 elseif (sgsize*(n+1)*(x+1) > 128) then
 gsize \leftarrow 128/(n+1)/(x+1)
 else
 gsize \leftarrow sgsz
 endif
 lgsize \leftarrow log(gsize)
 wsz \leftarrow 128/(x+1)
 vsz \leftarrow 128
 ds \leftarrow cs \leftarrow signed
 bs \leftarrow signed ^ m
 zs \leftarrow signed or m or n
 zsize \leftarrow gsize*(x+1)
 h \leftarrow (2*gsize) + log(vsz) - lgsize
 spos \leftarrow (a8..0) and (2*gsize-1)

1990

```
E.MUL.ADD.X:
  if (sgsize < 8) then
    gsize ← 8
  elseif (sgsize*(n+1)*(x+1) > 128) then
    gsize ← 128/(n+1)/(x+1)
  else
    gsize ← sgsz
  endif
  ds ← signed
  cs ← signed ^ m
  zs ← signed or m or n
  zsize ← gsize*(x+1)
  h ← (2*gsz) + n
  spos ← (a8..0) and (2*gsz-1)
endcase
dpos ← (0 || a23..16) and (zsize-1)
r ← spos
sfsz ← (0 || a31..24) and (zsize-1)
tfsz ← (sfsz = 0) or ((sfsz+dpos) > zsize) ? zsize-dpos : sfsz
fsz ← (tfsz + spos > h) ? h - spos : tfsz
if (b10..9 = Z) and not as then
  rnd ← F
else
  rnd ← b10..9
endif
```

FIG. 196 (CONTINUED)

1990

```

for k ← 0 to wsize-zsize by zsize
  i ← k*gsize/zsize
  case op of
    E.CON.X:
      q[0] ← 02*gsize+7*lgsize
      for j ← 0 to vsize-gsize by gsize
        if n then
          if (~i) & j & gsize = 0 then
            q[j+gsize] ← q[j] + mul(gsize,h,ms,m,i+128-j,bs,b,j)
          else
            q[j+gsize] ← q[j] - mul(gsize,h,ms,m,i+128-j+2*gsize,bs,b,j)
          endif
        else
          q[j+gsize] ← q[j] + mul(gsize,h,ms,m,i+128-j,bs,b,j)
        endif
      endfor
      p ← q[vsize]
    E.MUL.ADD.X:
      di ← ((ds and dk+zsize-1)h-zsize-r || (dk+zsize-1..k) || 0r)
      if n then
        if (i and gsize) = 0 then
          p ← mul(gsize,h,ds,d,i,cs,c,i)-
mul(gsize,h,ds,d,i+gsize,cs,c,i+gsize)+di
        else
          p ← mul(gsize,h,ds,d,i,cs,c,i+gsize)+mul(gsize,h,ds,d,i,cs,c,i+gsize)+di
        endif
      else
        p ← mul(gsize,h,ds,d,i,cs,c,i) + di
      endif
    endcase
  case rnd of
    N:
      s ← 0h-r || ~pr || prr-1
    Z:
      s ← 0h-r || ph-1r
    F:
      s ← 0h
    C:
      s ← 0h-r || 1r
  endcase
  v ← ((zs & ph-1) || p) + (0 || s)
  if (vh..r+fsize = (zs & vr+fsize-1)h+1-r-fsize) or not (l and (op = E.EXTRACT)) then
    w ← (zs & vr+fsize-1)zsize-fsize-dpos || vfsize-1+r..r || 0dpos
  else
    w ← (zs ? (vh || ~vhzsize-dpos-1) : 1zsize-dpos) || 0dpos
  endif
  Zzsize-1+k..k ← w
endfor
RegWrite(rd, 128, z)
enddef

```

FIG. 196 (CONTINUED)

2010

Operation codes

E.MUL.X	Ensemble multiply extract
E.EXTRACT	Ensemble extract
E.SCAL.ADD.X	Ensemble scale add extract

Format

E.op ra=rd,rc,rb

ra=eop(rd,rc,rb)

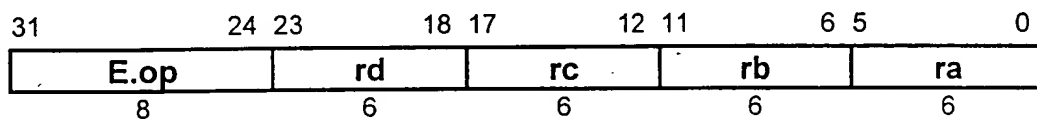


FIG. 20A

Figures 19B and 20B has blank fields: should be .

fsiz	dpos	x	s	n	m	l	rnd	gssp
------	------	---	---	---	---	---	-----	------

Fig. 20B

2020

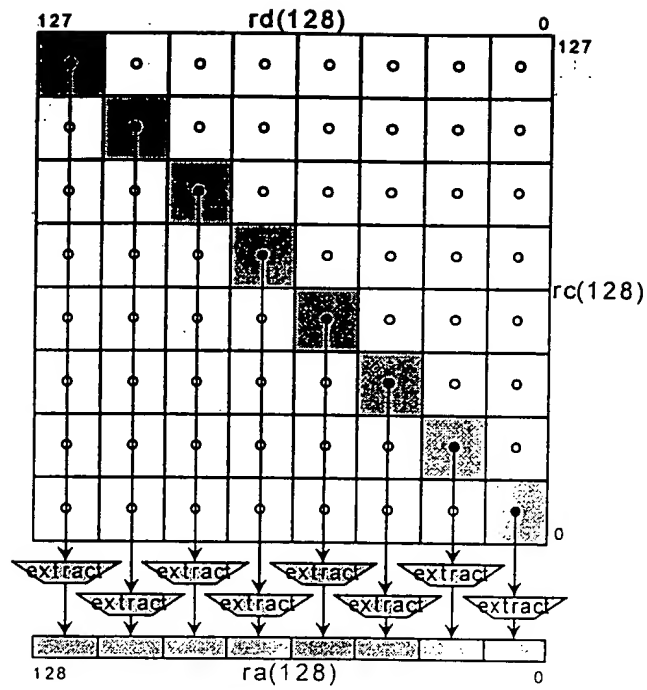
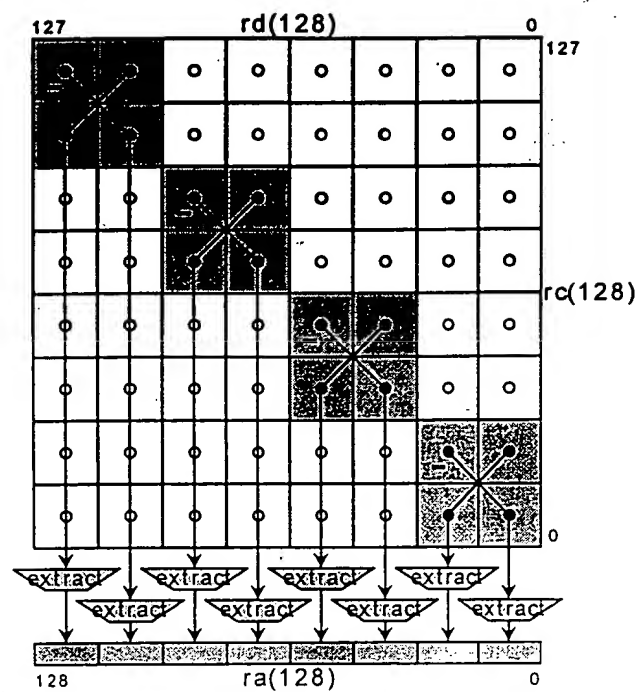


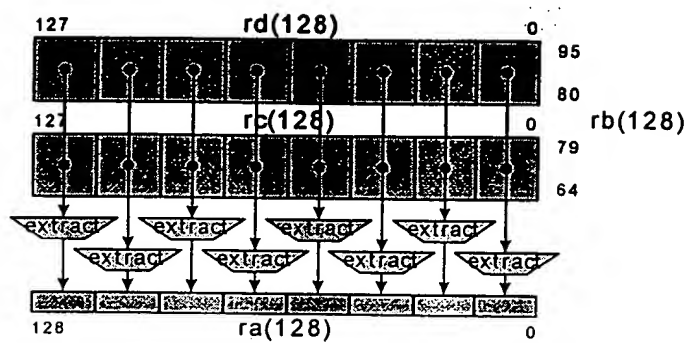
FIG. 20C



Ensemble complex multiply extract doublets

The ensemble-multiply-extract instructions (E.MUL.X), when the x bit is set, multiply the low-order 64 bits of each of the rc and rb registers and produce extended (double-size) results.

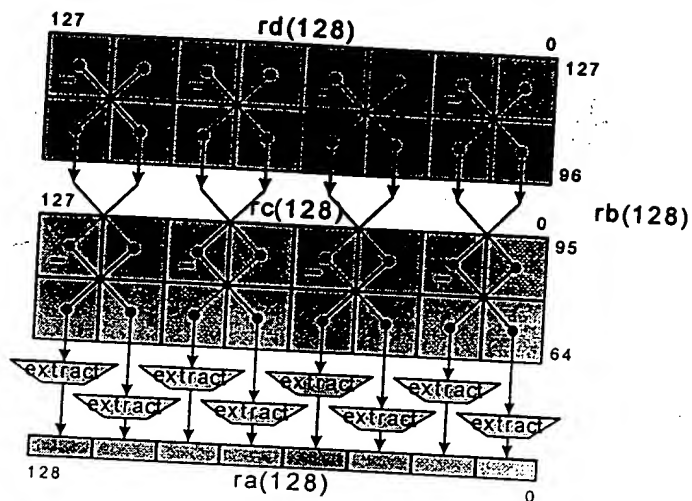
2040



Ensemble scale add extract doublets

FIG. 20E

2050

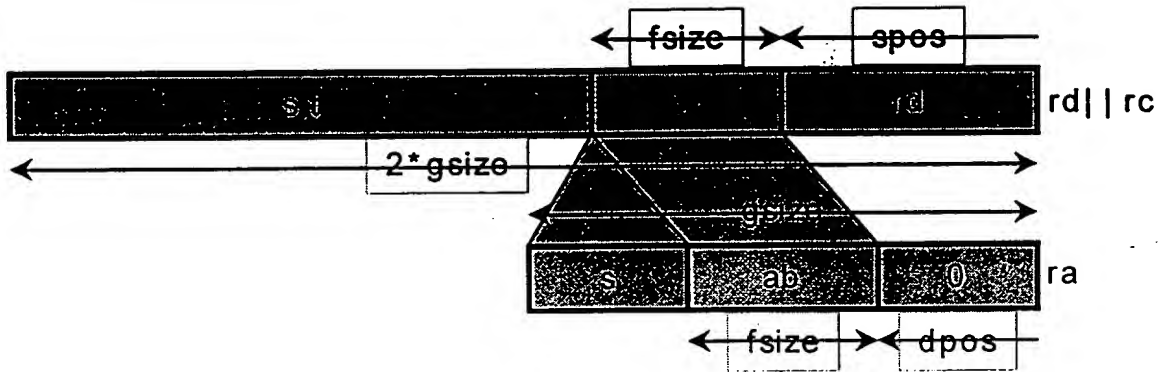


Ensemble complex scale add extract doublets

The ensemble-scale-add-extract instructions (E.SCLADD.X), when the x bit is set, multiply the low-order 64 bits of each of the rd and rc registers by the rb register fields and produce extended (double-size) results.

FIG. 20F

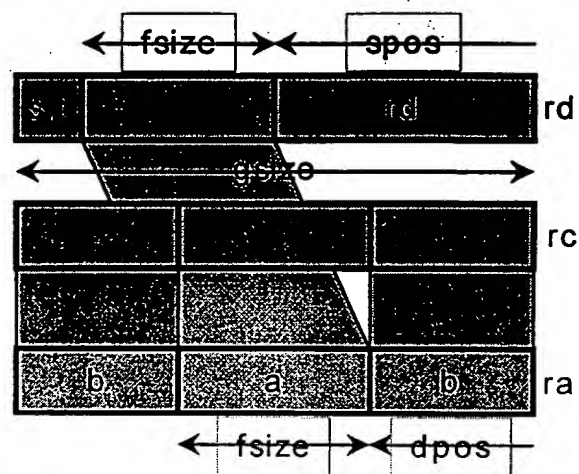
2060



Ensemble extract

FIG. 206

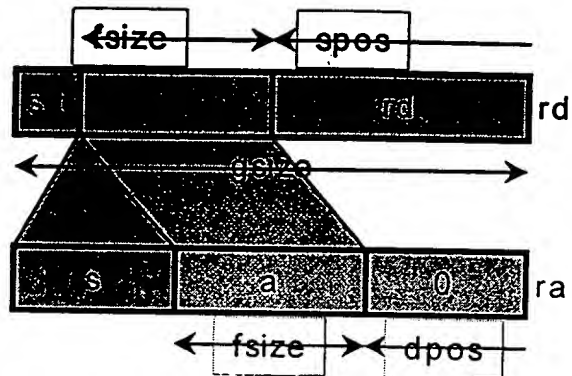
2070



Ensemble merge extract

FIG. 20H

2080



Ensemble expand extract

Fig. 201

← 2090

Definition

```
def mul(size,h,vs,v,i,ws,w,j) as
  mul ← ((vs&vsize-1+i)h-size || vsize-1+i..i) * ((ws&wsize-1+j)h-size || wsize-1+j..j)
enddef
```

```
def EnsembleExtract(op,ra,rb,rc,rd) as
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 128)
  b ← RegRead(rb, 128)
  case b8..0 of
    0..255:
      sgsz ← 128
    256..383:
      sgsz ← 64
    384..447:
      sgsz ← 32
    448..479:
      sgsz ← 16
    480..495:
      sgsz ← 8
    496..503:
      sgsz ← 4
    504..507:
      sgsz ← 2
    508..511:
      sgsz ← 1
  endcase
  l ← b11
  m ← b12
  n ← b13
  signed ← b14
  x ← b15
  case op of
    E.EXTRACT:
      gsize ← sgsz*(2-(m or x))
      zsize ← sgsz
      h ← gsize
      as ← signed
      spos ← (b8..0) and (gsz-1)
```

F16 20 J

2090

E.SCAL.ADD.X:

```

if (sgsize < 8) then
    gsize ← 8
elseif (sgsize*(n+1) > 32) then
    gsize ← 32/(n+1)
else
    gsize ← sgsz
endif
ds ← cs ← signed
bs ← signed ^ m
as ← signed or m or n
zsize ← gsize*(x+1)
h ← (2*gsz) + 1 + n
spos ← (b8..0) and (2*gsz-1)

```

E.MUL.X:

```

if (sgsize < 8) then
    gsize ← 8
elseif (sgsize*(n+1)*(x+1) > 128) then
    gsize ← 128/(n+1)/(x+1)
else
    gsize ← sgsz
endif
ds ← signed
cs ← signed ^ m
as ← signed or m or n
zsize ← gsize*(x+1)
h ← (2*gsz) + n
spos ← (b8..0) and (2*gsz-1)

```

endcase

dpos ← (0 || b23..16) and (zsize-1)

r ← spos

sfsz ← (0 || b31..24) and (zsize-1)

tfsz ← (sfsz = 0) or ((sfsz+dpos) > zsize) ? zsize-dpos : sfsz

fsize ← (tfsz + spos > h) ? h - spos : tfsz

if (b10..9 = Z) and not as then

 rnd ← F

else

 rnd ← b10..9

endif

F16.20J (CONTINUED)

2090

```

for j ← 0 to 128-zsize by zsize
  i ← j*gsize/zsize
  case op of
    E.EXTRACT:
      if m or x then
        p ← dgsize+i-1..i
      else
        p ← (d || c)gsize+i-1..i
      endif
    E.MUL.X:
      if n then
        if (i and gsize) = 0 then
          p ← mul(gsize,h,ds,d,i,cs,c,i)-
mul(gsize,h,ds,d,i+gsize,cs,c,i+gsize)
        else
          p ←
mul(gsize,h,ds,d,i,cs,c,i+gsize)+mul(gsize,h,ds,d,i,cs,c,i+gsize)
        endif
      else
        p ← mul(gsize,h,ds,d,i,cs,c,i)
      endif
    E.SCAL.ADD.X:
      if n then
        if (i and gsize) = 0 then
          p ← mul(gsize,h,ds,d,i,bs,b,64+2*gsize)
            + mul(gsize,h,cs,c,i,bs,b,64)
            - mul(gsize,h,ds,d,i+gsize,bs,b,64+3*gsize)
            - mul(gsize,h,cs,c,i+gsize,bs,b,64+gsize)
        else
          p ← mul(gsize,h,ds,d,i,bs,b,64+3*gsize)
            + mul(gsize,h,cs,c,i,bs,b,64+gsize)
            + mul(gsize,h,ds,d,i+gsize,bs,b,64+2*gsize)
            + mul(gsize,h,cs,c,i+gsize,bs,b,64)
        endif
      else
        p ← mul(gsize,h,ds,d,i,bs,b,64+gsize) + mul(gsize,h,cs,c,i,bs,b,64)
      endif
    endif
  endcase
case rnd of
  N:
    s ← 0h-r || ~pr || prr-1
  Z:
    s ← 0h-r || ph-1
  F:
    s ← 0h
  C:
    s ← 0h-r || 1r
endcase

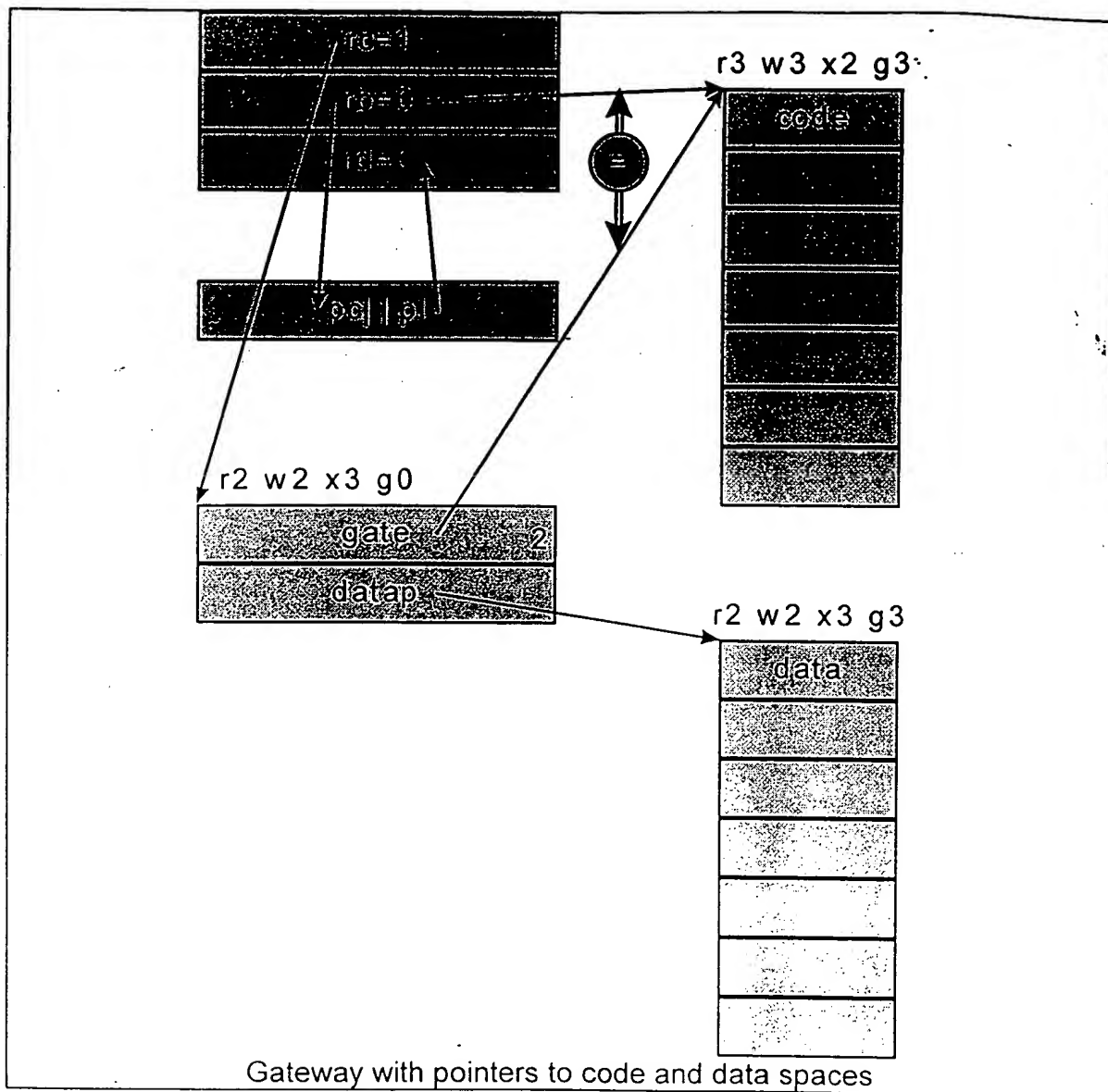
```

FIG. 20 (CONTINUED)

```

v ← ((as & ph-1) || p) + (0 || s)
if (vh..r+fsz = (as & vr+fsz-1)h+1-r-fsz) or not (l and (op = E.EXTRACT)) then
    w ← (as & vr+fsz-1)zsize-fsz-dpos || vfsz-1+r..r || 0dpos
else
    w ← (s ? (vh || ~vhzsize-dpos-1) : 1zsize-dpos) || 0dpos
endif
if m and (op = E.EXTRACT) then
    zzsize-1+j..j ← casize-1+j..dpos+fsz+j || wdpos+fsz-1..dpos || cdpos-1+j..j
else
    zzsize-1+j..j ← w
endif
endfor
RegWrite(ra, 128, z)
enddef

```



Typical dynamic-linked, inter-gateway calling sequence:

caller:

caller:	A.ADDI	sp@-size	// allocate caller stack frame
	S.I.64.A	lp,sp,off	
	S.I.64.A	dp,sp,off	
	...		
	L.I.64.A	lp=dp,off	// load lp
	L.I.64.A	dp=dp,off	// load dp
	B.GATE		
	L.I.64.A	dp,sp,off	
	... (code using dp)		
	L.I.64.A	lp=sp,off	// restore original lp register
	A.ADDI	sp=size	// deallocate caller stack frame
	B	lp	// return

callee (non-leaf):

callee:	L.I.64.A	dp=dp,off	// load dp with data pointer
	S.I.64.A	sp,dp,off	
	L.I.64.A	sp=dp,off	// new stack pointer
	S.I.64.A	lp,sp,off	
	S.I.64.A	dp,sp,off	
	... (using dp)		
	L.I.64.A	dp,sp,off	
	... (code using dp)		
	L.I.64.A	lp=sp,off	// restore original lp register
	L.I.64.A	sp=sp,off	// restore original sp register
	B.DOWN	lp	

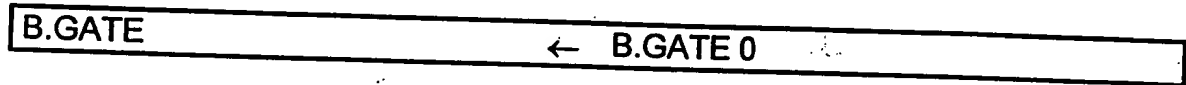
callee (leaf, no stack):

callee:	... (using dp)	
	B.DOWN	lp

Operation codes



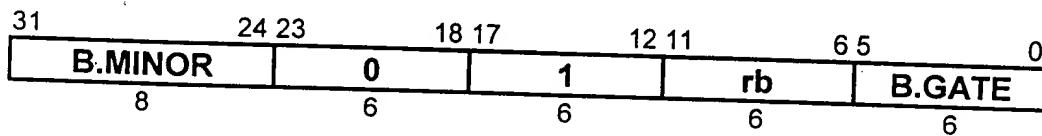
Equivalencies



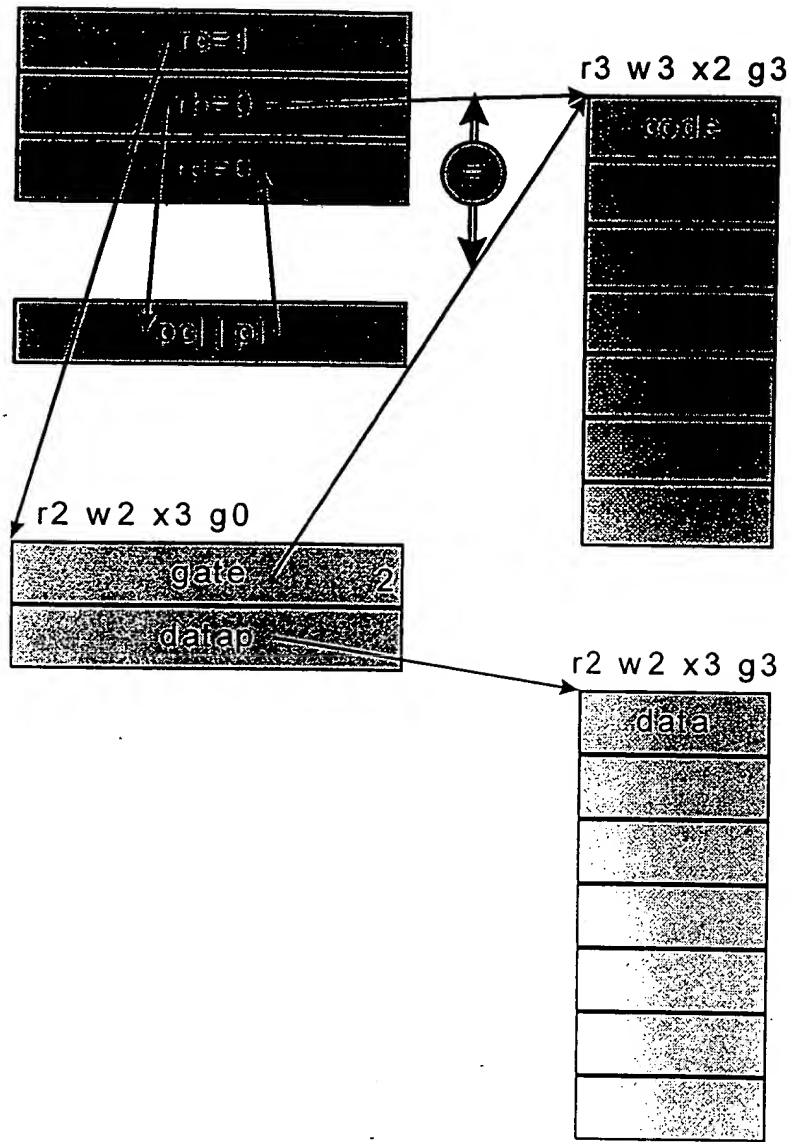
Format

B.GATE rb

bgate(rb)



← 2170



Branch gateway

F16. 210

← 2190

Definition

```
def BranchGateway(rd,rc,rb) as
  c ← RegRead(rc, 64)
  b ← RegRead(rb, 64)
  if (rd ≠ 0) or (rc ≠ 1) then
    raise ReservedInstruction
  endif
  if c2..0 ≠ 0 then
    raise AccessDisallowedByVirtualAddress
  endif
  d ← ProgramCounter63..2+1 || PrivilegeLevel
  if PrivilegeLevel < b1..0 then
    m ← LoadMemoryG(c,c,64,L)
    if b ≠ m then
      raise GatewayDisallowed
    endif
    PrivilegeLevel ← b1..0
  endif
  ProgramCounter ← b63..2 || 02
  RegWrite(rd, 64, d)
  raise TakenBranch
enddef
```

Fig. 21E

Exceptions

Reserved Instruction
Gateway disallowed
Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

2210

Operation codes

E.SCAL.ADD.F.16	Ensemble scale add floating-point half
E.SCAL.ADD.F.32	Ensemble scale add floating-point single
E.SCAL.ADD.F.64	Ensemble scale add floating-point double

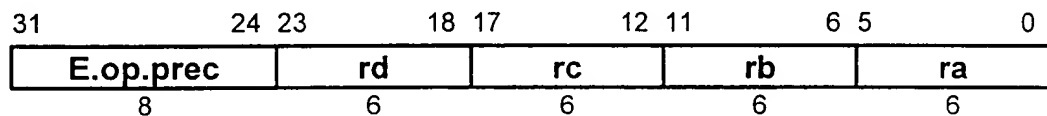
Selection

class	op	prec		
scale add	E.SCAL.ADD.F	16	32	64

Format

E.op.prec ra=rd,rc,rb

ra=eopprec(rd,rc,rb)



F16. 22A

223°

Definition

```
def EnsembleFloatingPointTernary(op,prec,rd,rc,rb,ra) as
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 128)
  b ← RegRead(rb, 128)
  for i ← 0 to 128-prec by prec
    di ← F(prec,di+prec-1..i)
    ci ← F(prec,ci+prec-1..i)
    ai ← fadd(fmul(di, F(prec,bprec-1..0)), fmul(ci, F(prec,b2*prec-1..prec)))
    ai+prec-1..i ← PackF(prec, ai, none)
  endfor
  RegWrite(ra, 128, a)
enddef
```

Fig. 22B

231

Operation codes

G.BOOLEAN	Group boolean
-----------	---------------

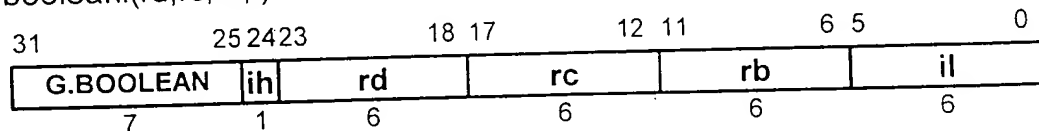
Selection

operation	function (binary)	function (decimal)
d	11110000	240
c	11001100	204
b	10101010	176
d&c&b	10000000	128
(d&c) b	11101010	234
d c b	11111110	254
d?c:b	11001010	202
d^c^b	10010110	150
~d^c^b	01101001	105
0	00000000	0

Format

G.BOOLEAN rd@trc,rb,f

rd=gbooleani(rd,rc,rb,f)



F16. 23A

2320

```

if f6=f5 then
  if f2=f1 then
    if f2 then
      rc ← max(trc,trb)
      rb ← min(trc,trb)
    else
      rc ← min(trc,trb)
      rb ← max(trc,trb)
    endif
    ih ← 0
    il ← 0 || f6 || f7 || f4 || f3 || f0
  else
    if f2 then
      rc ← trb
      rb ← trc
    else
      rc ← trc
      rb ← trb
    endif
    ih ← 0
    il ← 1 || f6 || f7 || f4 || f3 || f0
  endif
else
  ih ← 1
  if f6 then
    rc ← trb
    rb ← trc
    il ← f1 || f2 || f7 || f4 || f3 || f0
  else
    rc ← trc
    rb ← trb
    il ← f2 || f1 || f7 || f4 || f3 || f0
  endif
endif
endif

```

F16. 236

2330

Definition

```
def GroupBoolean (ih,rd,rc,rb,il)
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 128)
  b ← RegRead(rb, 128)
  if ih=0 then
    if il5=0 then
      f ← il3 || il4 || il4 || il2 || il1 || (rc>rb)2 || il0
    else
      f ← il3 || il4 || il4 || il2 || il1 || 0 || 1 || il0
    endif
  else
    f ← il3 || 0 || 1 || il2 || il1 || il5 || il4 || il0
  endif
  for i ← 0 to 127 by size
    ai ← f(di||ci||bi)
  endfor
  RegWrite(rd, 128, a)
enddef
```

F16 23C

← 2410

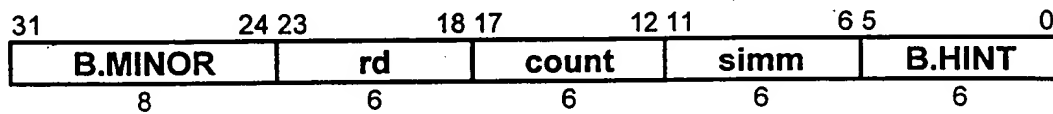
Operation codes

B.HINT	Branch Hint
--------	-------------

Format

B.HINT badd,count,rd

bhint(badd,count,rd)



sim ← badd-pc-4

FIG. 24A

2430

Definition

```
def BranchHint(rd,count,simm) as
  d ← RegRead(rd, 64)
  if (d1..0) ≠ 0 then
    raise AccessDisallowedByVirtualAddress
  endif
  FetchHint(ProgramCounter + 4 + (0 || simm || 02), d63..2 || 02, count)
enddef
```

F16.24B

2460

Exceptions

Access disallowed by virtual address

F16.24C

2510

Operation codes

E.SINK.F.16	Ensemble convert floating-point doublets from half nearest default
E.SINK.F.16.C	Ensemble convert floating-point doublets from half ceiling
E.SINK.F.16.C.D	Ensemble convert floating-point doublets from half ceiling default
E.SINK.F.16.F	Ensemble convert floating-point doublets from half floor
E.SINK.F.16.F.D	Ensemble convert floating-point doublets from half floor default
E.SINK.F.16.N	Ensemble convert floating-point doublets from half nearest
E.SINK.F.16.X	Ensemble convert floating-point doublets from half exact
E.SINK.F.16.Z	Ensemble convert floating-point doublets from half zero
E.SINK.F.16.Z.D	Ensemble convert floating-point doublets from half zero default
E.SINK.F.32	Ensemble convert floating-point quadlets from single nearest default
E.SINK.F.32.C	Ensemble convert floating-point quadlets from single ceiling
E.SINK.F.32.C.D	Ensemble convert floating-point quadlets from single ceiling default
E.SINK.F.32.F	Ensemble convert floating-point quadlets from single floor
E.SINK.F.32.F.D	Ensemble convert floating-point quadlets from single floor default
E.SINK.F.32.N	Ensemble convert floating-point quadlets from single nearest
E.SINK.F.32.X	Ensemble convert floating-point quadlets from single exact
E.SINK.F.32.Z	Ensemble convert floating-point quadlets from single zero
E.SINK.F.32.Z.D	Ensemble convert floating-point quadlets from single zero default
E.SINK.F.64	Ensemble convert floating-point octlets from double nearest default
E.SINK.F.64.C	Ensemble convert floating-point octlets from double ceiling
E.SINK.F.64.C.D	Ensemble convert floating-point octlets from double ceiling default
E.SINK.F.64.F	Ensemble convert floating-point octlets from double floor
E.SINK.F.64.F.D	Ensemble convert floating-point octlets from double floor default
E.SINK.F.64.N	Ensemble convert floating-point octlets from double nearest
E.SINK.F.64.X	Ensemble convert floating-point octlets from double exact
E.SINK.F.64.Z	Ensemble convert floating-point octlets from double zero
E.SINK.F.64.Z.D	Ensemble convert floating-point octlets from double zero default
E.SINK.F.128	Ensemble convert floating-point hexlet from quad nearest default
E.SINK.F.128.C	Ensemble convert floating-point hexlet from quad ceiling
E.SINK.F.128.C.D	Ensemble convert floating-point hexlet from quad ceiling default
E.SINK.F.128.F	Ensemble convert floating-point hexlet from quad floor
E.SINK.F.128.F.D	Ensemble convert floating-point hexlet from quad floor default
E.SINK.F.128.N	Ensemble convert floating-point hexlet from quad nearest
E.SINK.F.128.X	Ensemble convert floating-point hexlet from quad exact
E.SINK.F.128.Z	Ensemble convert floating-point hexlet from quad zero
E.SINK.F.128.Z.D	Ensemble convert floating-point hexlet from quad zero default

FIG. 25A

2510

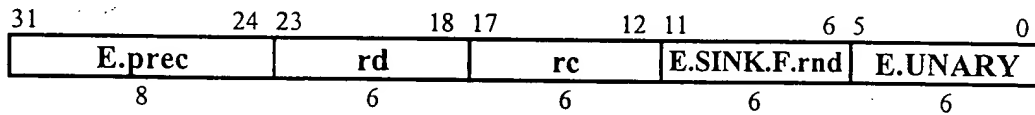
Selection

	op	prec	round/trap
integer from float	SINK.	16 32 64 128	NONE C F N X Z C.D F.D Z.D

Format

E.SINK.F.prec.rnd rd=rc

rd=esinkfprecrnd(rc)



F16.23A CONTINUED

← 2530

Definition

```
def EnsembleSinkFloatingPoint(prec,round,rd,rc) as
  c ← RegRead(rc, 128)
  for i ← 0 to 128-prec by prec
    ci ← F(prec,ci+prec-1..i)
    ai+prec-1..i ← fsinkr(prec, ci, round)
  endfor
  RegWrite[rd, 128, a]
enddef
```

6-11-14-6

Definition

```

def eb ← ebits(prec) as
  case pref of
    16:
      eb ← 5
    32:
      eb ← 8
    64:
      eb ← 11
    128:
      eb ← 15
  endcase
enddef

```

```

def eb ← ebias(prec) as
  eb ← 0 || 1ebits(prec)-1
enddef

```

```

def fb ← fbits(prec) as
  fb ← prec - 1 - eb
enddef

```

```

def a ← F(prec, ai) as
  a.s ← aiprec-1
  ae ← aiprec-2..fbits(prec)
  af ← aifbits(prec)-1..0
  if ae = 1ebits(prec) then
    if af = 0 then
      a.t ← INFINITY
    elseif affbits(prec)-1 then
      a.t ← SNaN
      a.e ← -fbits(prec)
      a.f ← 1 || affbits(prec)-2..0
    else
      a.t ← QNaN
      a.e ← -fbits(prec)
      a.f ← af
    endif
  elseif ae = 0 then
    if af = 0 then
      a.t ← ZERO
    else
      a.t ← NORM
      a.e ← 1-ebias(prec)-fbits(prec)
      a.f ← 0 || af
    endif
  else
    a.t ← NORM
    a.e ← ae-ebias(prec)-fbits(prec)
    a.f ← 1 || af
  endif
enddef

```

2570

F16. 25 D

✓ 2576

```
def a ← DEFAULTQNaN as
  a.s ← 0
  a.t ← QNaN
  a.e ← -1
  a.f ← 1
enddef
```

```
def a ← DEFAULTSNaN as
  a.s ← 0
  a.t ← SNaN
  a.e ← -1
  a.f ← 1
enddef
```

```
def fadd(a,b) as faddr(a,b,N) enddef
```

```
def c ← faddr(a,b,round) as
  if a.t=NORM and b.t=NORM then
    // d,e are a,b with exponent aligned and fraction adjusted
    if a.e > b.e then
      d ← a
      e.t ← b.t
      e.s ← b.s
      e.e ← a.e
      e.f ← b.f || 0a.e-b.e
    else if a.e < b.e then
      d.t ← a.t
      d.s ← a.s
      d.e ← b.e
      d.f ← a.f || 0b.e-a.e
      e ← b
    endif
    c.t ← d.t
    c.e ← d.e
    if d.s = e.s then
      c.s ← d.s
      c.f ← d.f + e.f
    elseif d.f > e.f then
      c.s ← d.s
      c.f ← d.f - e.f
    elseif d.f < e.f then
      c.s ← e.s
      c.f ← e.f - d.f
    else
      c.s ← r=F
      c.t ← ZERO
    endif
  endif
```

616.7-1

2570

```
// priority is given to b operand for NaN propagation
elseif (b.t=SNAN) or (b.t=QNAN) then
    c ← b
elseif (a.t=SNAN) or (a.t=QNAN) then
    c ← a
elseif a.t=ZERO and b.t=ZERO then
    c.t ← ZERO
    c.s ← (a.s and b.s) or (round=F and (a.s or b.s))
// NULL values are like zero, but do not combine with ZERO to alter sign
elseif a.t=ZERO or a.t=NULL then
    c ← b
elseif b.t=ZERO or b.t=NULL then
    c ← a
elseif a.t=INFINITY and b.t=INFINITY then
    if a.s ≠ b.s then
        c ← DEFAULTSNAN // Invalid
    else
        c ← a
    endif
elseif a.t=INFINITY then
    c ← a
elseif b.t=INFINITY then
    c ← b
else
    assert FALSE // should have covered all the cases above
endif
enddef

def b ← fneg(a) as
    b.s ← ~a.s
    b.t ← a.t
    b.e ← a.e
    b.f ← a.f
enddef

def fsub(a,b) as fsubr(a,b,N) enddef

def fsubr(a,b,round) as faddr(a,fneg(b),round) enddef

def frsub(a,b) as frsubr(a,b,N) enddef

def frsubr(a,b,round) as faddr(fneg(a),b,round) enddef
```

1 - 1 D 1000 - 100

2570

```

def c ← fcom(a,b) as
  if (a.t=SNAN) or (a.t=QNAN) or (b.t=SNAN) or (b.t=QNAN) then
    c ← U
  elseif a.t=INFINITY and b.t=INFINITY then
    if a.s ≠ b.s then
      c ← (a.s=0) ? G: L
    else
      c ← E
    endif
  elseif a.t=INFINITY then
    c ← (a.s=0) ? G: L
  elseif b.t=INFINITY then
    c ← (b.s=0) ? G: L
  elseif a.t=NORM and b.t=NORM then
    if a.s ≠ b.s then
      c ← (a.s=0) ? G: L
    else
      if a.e > b.e then
        af ← a.f
        bf ← b.f || 0a.e-b.e
      else
        af ← a.f || 0b.e-a.e
        bf ← b.f
      endif
      if af = bf then
        c ← E
      else
        c ← ((a.s=0) ^ (af > bf)) ? G : L
      endif
    endif
  elseif a.t=NORM then
    c ← (a.s=0) ? G: L
  elseif b.t=NORM then
    c ← (b.s=0) ? G: L
  elseif a.t=ZERO and b.t=ZERO then
    c ← E
  else
    assert FALSE // should have covered al the cases above
  endif
enddef

```

2570

```

def c ← fmul(a,b) as
  if a.t=NORM and b.t=NORM then
    c.s ← a.s ^ b.s
    c.t ← NORM
    c.e ← a.e + b.e
    c.f ← a.f * b.f
    // priority is given to b operand for NaN propagation
  elseif (b.t=SNAN) or (b.t=QNAN) then
    c.s ← a.s ^ b.s
    c.t ← b.t
    c.e ← b.e
    c.f ← b.f
  elseif (a.t=SNAN) or (a.t=QNAN) then
    c.s ← a.s ^ b.s
    c.t ← a.t
    c.e ← a.e
    c.f ← a.f
  elseif a.t=ZERO and b.t=INFINITY then
    c ← DEFAULTSNAN // Invalid
  elseif a.t=INFINITY and b.t=ZERO then
    c ← DEFAULTSNAN // Invalid
  elseif a.t=ZERO or b.t=ZERO then
    c.s ← a.s ^ b.s
    c.t ← ZERO
  else
    assert FALSE // should have covered all the cases above
  endif
enddef

```

730 (10-11-12)

✓ 2570

```

def c ← fdivr(a,b) as
  if a.t=NORM and b.t=NORM then
    c.s ← a.s ^ b.s
    c.t ← NORM
    c.e ← a.e - b.e + 256
    c.f ← (a.f || 0256) / b.f
  // priority is given to b operand for NaN propagation
  elseif (b.t=SNAN) or (b.t=QNAN) then
    c.s ← a.s ^ b.s
    c.t ← b.t
    c.e ← b.e
    c.f ← b.f
  elseif (a.t=SNAN) or (a.t=QNAN) then
    c.s ← a.s ^ b.s
    c.t ← a.t
    c.e ← a.e
    c.f ← a.f
  elseif a.t=ZERO and b.t=ZERO then
    c ← DEFAULTSNAN // Invalid
  elseif a.t=INFINITY and b.t=INFINITY then
    c ← DEFAULTSNAN // Invalid
  elseif a.t=ZERO then
    c.s ← a.s ^ b.s
    c.t ← ZERO
  elseif a.t=INFINITY then
    c.s ← a.s ^ b.s
    c.t ← INFINITY
  else
    assert FALSE // should have covered al the cases above
  endif
enddef

```

```

def msb ← findmsb(a) as
  MAXF ← 218 // Largest possible f value after matrix multiply
  for j ← 0 to MAXF
    if aMAXF-1..j = (0MAXF-1-j || 1) then
      msb ← j
    endif
  endfor
enddef

```

find msb Def

2570

```

def ai ← PackF(prec,a,round) as
  case a.t of
    NORM:
      msb ← findmsb(a.f)
      rn ← msb-1-fbits(prec) // lsb for normal
      rdn ← -ebias(prec)-a.e-1-fbits(prec) // lsb if a denormal
      rb ← (rn > rdn) ? rn : rdn
      if rb ≤ 0 then
        aifr ← a.fmsb-1..0 || 0rb
        eadj ← 0
      else
        case round of
          C:
            s ← 0msb-rb || (~a.s)rb
          F:
            s ← 0msb-rb || (a.s)rb
          N, NONE:
            s ← 0msb-rb || ~a.frb || a.frbb-1
          X:
            if a.frb-1..0 ≠ 0 then
              raise FloatingPointArithmetic // Inexact
            endif
            s ← 0
          Z:
            s ← 0
        endcase
      v ← (0 || a.fmsb..0) + (0 || s)
      if vmsb = 1 then
        aifr ← vmsb-1..rb
        eadj ← 0
      else
        aifr ← 0fbits(prec)
        eadj ← 1
      endif
    endif
    aien ← a.e + msb - 1 + eadj + ebias(prec)
    if aien ≤ 0 then
      if round = NONE then
        ai ← a.s || 0ebits(prec) || aifr
      else
        raise FloatingPointArithmetic // Underflow
      endif
    elseif aien ≥ 1ebits(prec) then
      if round = NONE then
        //default: round-to-nearest overflow handling
        ai ← a.s || 1ebits(prec) || 0fbits(prec)
      else
        raise FloatingPointArithmetic // Overflow
      endif
    else
      ai ← a.s || aienebits(prec)-1..0 || aifr
    endif
  endif

```

Fig. 2 (continued)

2570

```
SNAN:
  if round  $\neq$  NONE then
    raise FloatingPointArithmetic //Invalid
  endif
  if  $-a.e < fbits(prec)$  then
     $ai \leftarrow a.s \parallel 1^{ebits(prec)} \parallel a.f_{a.e-1..0} \parallel 0^{fbits(prec)+a.e}$ 
  else
     $lsb \leftarrow a.f_{a.e-1-fbits(prec)+1..0} \neq 0$ 
     $ai \leftarrow a.s \parallel 1^{ebits(prec)} \parallel a.f_{a.e-1..-a.e-1-fbits(prec)+2} \parallel lsb$ 
  endif
QNaN:
  if  $-a.e < fbits(prec)$  then
     $ai \leftarrow a.s \parallel 1^{ebits(prec)} \parallel a.f_{a.e-1..0} \parallel 0^{fbits(prec)+a.e}$ 
  else
     $lsb \leftarrow a.f_{a.e-1-fbits(prec)+1..0} \neq 0$ 
     $ai \leftarrow a.s \parallel 1^{ebits(prec)} \parallel a.f_{a.e-1..-a.e-1-fbits(prec)+2} \parallel lsb$ 
  endif
ZERO:
   $ai \leftarrow a.s \parallel 0^{ebits(prec)} \parallel 0^{fbits(prec)}$ 
INFINITY:
   $ai \leftarrow a.s \parallel 1^{ebits(prec)} \parallel 0^{fbits(prec)}$ 
endcase
defdef
```

FIG. 24D

4820

```

def ai ← fsinkr(prec, a, round) as
  case a.t of
    NORM:
      msb ← findmsb(a.f)
      rb ← -a.e
      if rb ≤ 0 then
        aifr ← a.fmsb..0 || 0-rb
        aims ← msb - rb
      else
        case round of
          C, C.D:
            s ← 0msb-rb || (~ai.s)rb
          F, F.D:
            s ← 0msb-rb || (ai.s)rb
          N, NONE:
            s ← 0msb-rb || ~ai.frb || ai.frb-1
          X:
            if ai.frb-1..0 ≠ 0 then
              raise FloatingPointArithmetic // Inexact
            endif
            s ← 0
          Z, Z.D:
            s ← 0
        endcase
      v ← (0 || a.fmsb..0) + (0 || s)
      if vmsb = 1 then
        aims ← msb + 1 - rb
      else
        aims ← msb - rb
      endif
      aifr ← vaims..rb
    endif
    if aims > prec then
      case round of
        C.D, F.D, NONE, Z.D:
          ai ← a.s || (~as)prec-1
        C, F, N, X, Z:
          raise FloatingPointArithmetic // Overflow
      endcase
    elseif a.s = 0 then
      ai ← aifr
    else
      ai ← -aifr
    endif
  ZERO:
    ai ← 0prec
  SNAN, QNAN:
    case round of
      C.D, F.D, NONE, Z.D:
        ai ← 0prec
      C, F, N, X, Z:
        raise FloatingPointArithmetic // Invalid
    endcase
enddef

```

4820

2575

```

endcase
INFINITY:
  case round of
    C.D, F.D, NONE, Z.D:
      ai ← a.s || (~as)prec-1
    C, F, N, X, Z:
      raise FloatingPointArithmetic // Invalid
  endcase
endcase
enddef

```

```

def c ← frecrest(a) as
  b.s ← 0
  b.t ← NORM
  b.e ← 0
  b.f ← 1
  c ← fest(fdiv(b,a))
enddef

```

```

def c ← frsqrest(a) as
  b.s ← 0
  b.t ← NORM
  b.e ← 0
  b.f ← 1
  c ← fest(fsqr(fdiv(b,a)))
enddef

```

```

def c ← fest(a) as
  if (a.t=NORM) then
    msb ← findmsb(a.f)
    a.e ← a.e + msb - 13
    a.f ← a.f.msb..msb-12 || 1
  else
    c ← a
  endif
enddef

```

```

def c ← fsqr(a) as
  if (a.t=NORM) and (a.s=0) then
    c.s ← 0
    c.t ← NORM
    if (a.e0 = 1) then
      c.e ← (a.e-127) / 2
      c.f ← sqr(a.f || 0127)
    else
      c.e ← (a.e-128) / 2
      c.f ← sqr(a.f || 0128)
    endif
  elseif (a.t=SNAN) or (a.t=QNAN) or a.t=ZERO or ((a.t=INFINITY) and (a.s=0)) then
    c ← a
  elseif ((a.t=NORM) or (a.t=INFINITY)) and (a.s=1) then
    c ← DEFAULTSNAN // Invalid
  else
    assert FALSE // should have covered all the cases above
  endif
enddef

```

Format

G.op.size rd=rc,rb

rd=gopsize(rc,rb)

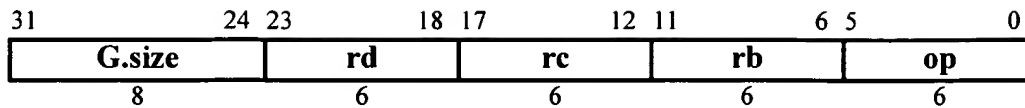


Fig. 26B

Operation codes

G.ADD.8	Group add bytes
G.ADD.16	Group add doublets
G.ADD.32	Group add quadlets
G.ADD.64	Group add octlets
G.ADD.128	Group add hexlet
G.ADD.L.8	Group add limit signed bytes
G.ADD.L.16	Group add limit signed doublets
G.ADD.L.32	Group add limit signed quadlets
G.ADD.L.64	Group add limit signed octlets
G.ADD.L.128	Group add limit signed hexlet
G.ADD.L.U.8	Group add limit unsigned bytes
G.ADD.L.U.16	Group add limit unsigned doublets
G.ADD.L.U.32	Group add limit unsigned quadlets
G.ADD.L.U.64	Group add limit unsigned octlets
G.ADD.L.U.128	Group add limit unsigned hexlet
G.ADD.8.O	Group add signed bytes check overflow
G.ADD.16.O	Group add signed doublets check overflow
G.ADD.32.O	Group add signed quadlets check overflow
G.ADD.64.O	Group add signed octlets check overflow
G.ADD.128.O	Group add signed hexlet check overflow
G.ADD.U.8.O	Group add unsigned bytes check overflow
G.ADD.U.16.O	Group add unsigned doublets check overflow
G.ADD.U.32.O	Group add unsigned quadlets check overflow
G.ADD.U.64.O	Group add unsigned octlets check overflow
G.ADD.U.128.O	Group add unsigned hexlet check overflow

Fig. 26A

Definition

```

def Crossbar(op,size,rd,rc,rb)
  c ← RegRead(rc, 128)
  b ← RegRead(rb, 128)
  shift ← b and (size-1)
  case op5..2 || 02 of
    X.COMPRESS:
      hsize ← size/2
      for i ← 0 to 64-hsize by hsize
        if shift ≤ hsize then
          ai+hsize-1..i ← ci+i+shift+hsize-1..i+shift
        else
          ai+hsize-1..i ← cshift-hsize..i+hsize-1shift-hsize || ci+i+size-1..i+shift
        endif
      endfor
      a127..64 ← 0
    X.COMPRESS.U:
      hsize ← size/2
      for i ← 0 to 64-hsize by hsize
        if shift ≤ hsize then
          ai+hsize-1..i ← ci+i+shift+hsize-1..i+shift
        else
          ai+hsize-1..i ← 0shift-hsize || ci+i+size-1..i+shift
        endif
      endfor
      a127..64 ← 0
    X.EXPAND:
      hsize ← size/2
      for i ← 0 to 64-hsize by hsize
        if shift ≤ hsize then
          ai+i+size-1..i ← ci+hsize-1..i+size-1hsize-shift || ci+hsize-1..i || 0shift
        else
          ai+i+size-1..i ← ci+size-shift-1..i || 0shift
        endif
      endfor
    X.EXPAND.U:
      hsize ← size/2
      for i ← 0 to 64-hsize by hsize
        if shift ≤ hsize then
          ai+i+size-1..i ← 0hsize-shift || ci+hsize-1..i || 0shift
        else
          ai+i+size-1..i ← ci+size-shift-1..i || 0shift
        endif
      endfor
    X.ROTL:
      for i ← 0 to 128-size by size
        ai+size-1..i ← ci+size-1-shift..i || ci+size-1..i+size-1-shift
      endfor

```

Fig. 32C

```

X.ROTR:
  for i ← 0 to 128-size by size
     $a_{i+size-1..i} \leftarrow c_{i+shift-1..i} \parallel c_{i+size-1..i+shift}$ 
  endfor
X.SHL:
  for i ← 0 to 128-size by size
     $a_{i+size-1..i} \leftarrow c_{i+size-1-shift..i} \parallel 0^{shift}$ 
  endfor
X.SHL.O:
  for i ← 0 to 128-size by size
    if  $c_{i+size-1..i+size-1-shift} \neq c_{i+size-1-shift}^{shift+1}$  then
      raise FixedPointArithmetic
    endif
     $a_{i+size-1..i} \leftarrow c_{i+size-1-shift..i} \parallel 0^{shift}$ 
  endfor
X.SHL.U.O:
  for i ← 0 to 128-size by size
    if  $c_{i+size-1..i+size-shift} \neq 0^{shift}$  then
      raise FixedPointArithmetic
    endif
     $a_{i+size-1..i} \leftarrow c_{i+size-1-shift..i} \parallel 0^{shift}$ 
  endfor
X.SHR:
  for i ← 0 to 128-size by size
     $a_{i+size-1..i} \leftarrow c_{i+size-1}^{shift} \parallel c_{i+size-1..i+shift}$ 
  endfor
X.SHR.U:
  for i ← 0 to 128-size by size
     $a_{i+size-1..i} \leftarrow 0^{shift} \parallel c_{i+size-1..i+shift}$ 
  endfor
endcase
RegWrite(rd, 128, a)
enddef

```

Fig. 32C (cont'd)

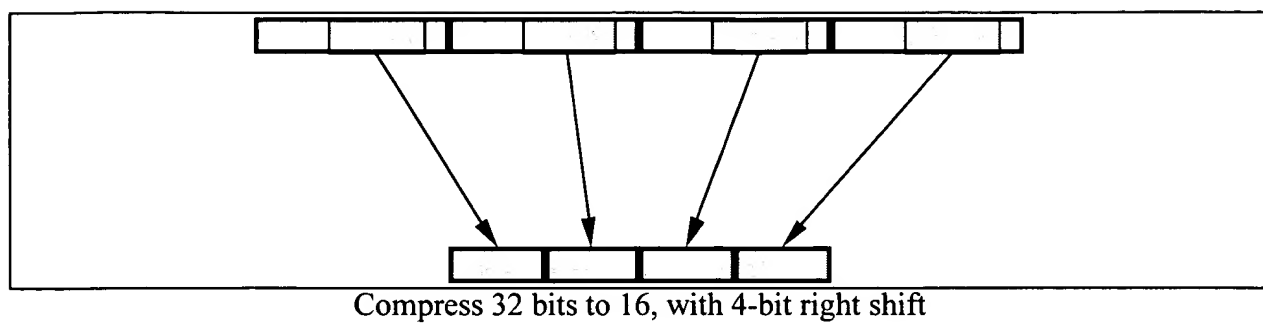


Fig. 32D

Format

X.EXTRACT ra=rd,rc,rb

ra=xextract(rd,rc,rb)

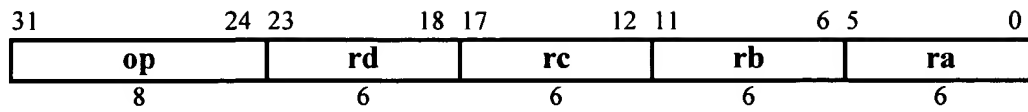


Fig. 33A

Fig. 33B

Definition

```
def CrossbarExtract(op,ra,rb,rc,rd) as
  d ← RegRead(rd, 128)
  c ← RegRead(rc, 128)
  b ← RegRead(rb, 128)
  case b8..0 of
    0..255:
      gsize ← 128
    256..383:
      gsize ← 64
    384..447:
      gsize ← 32
    448..479:
      gsize ← 16
    480..495:
      gsize ← 8
    496..503:
      gsize ← 4
    504..507:
      gsize ← 2
    508..511:
      gsize ← 1
  endcase
  m ← b12
  as ← signed ← b14
  h ← (2-m)*gsiz
  spos ← (b8..0) and ((2-m)*gsiz-1)
  dpos ← (0 || b23..16) and (gsiz-1)
  sfsiz ← (0 || b31..24) and (gsiz-1)
  tfsiz ← (sfsiz = 0) or ((sfsiz+dpos) > gsiz) ? gsiz-dpos : sfsiz
  fsiz ← (tfsiz + spos > h) ? h - spos : tfsiz
  for i ← 0 to 128-gsiz by gsiz
    case op of
      X.EXTRACT:
        if m then
          p ← dgsiz+i-1..i
        else
          p ← (d || c)2*(gsiz+i)-1..2*i
        endif
      endcase
      v ← (as & ph-1) || p
      w ← (as & vspos+fsiz-1)gsiz-fsiz-dpos || vfsiz-1+spos..spos || 0dpos
      if m then
        asize-1+i..i ← cgsiz-1+i..dpos+fsiz+i || wdpos+fsiz-1..dpos || cdpos-1+1..i
      else
        asize-1+i..i ← w
      endif
    endfor
  RegWrite(ra, 128, a)
enddef
```

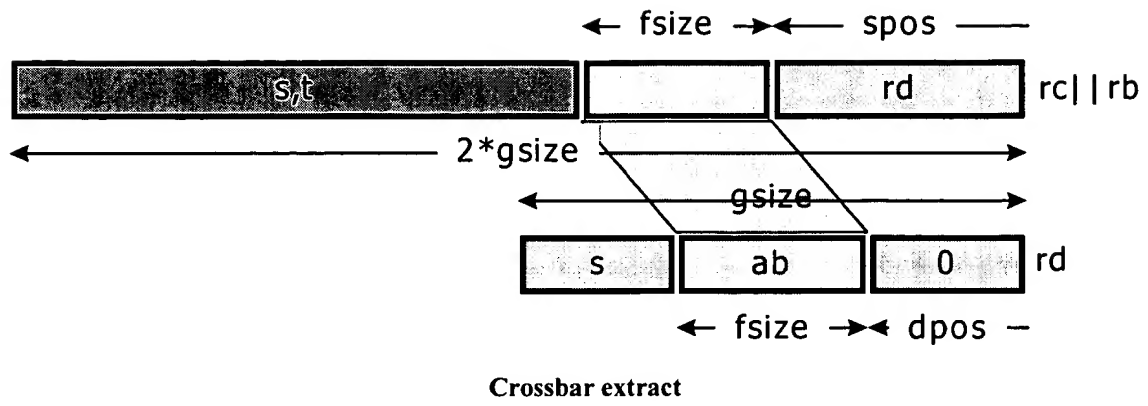


Fig. 33C

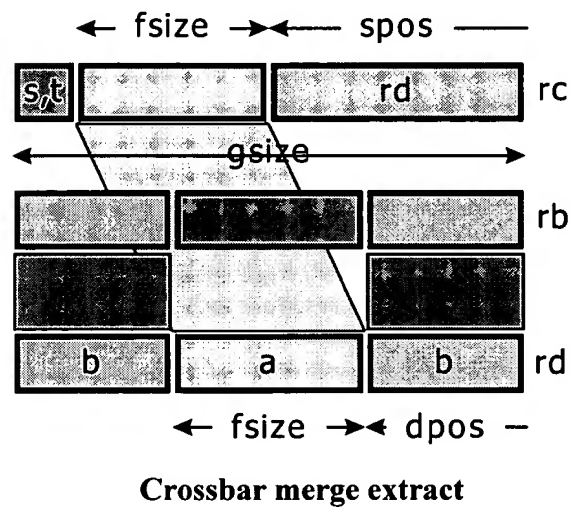


Fig. 33D

X.SHUFFLE.4	Crossbar shuffle within pecks
X.SHUFFLE.8	Crossbar shuffle within bytes
X.SHUFFLE.16	Crossbar shuffle within doublets
X.SHUFFLE.32	Crossbar shuffle within quadlets
X.SHUFFLE.64	Crossbar shuffle within octlets
X.SHUFFLE.128	Crossbar shuffle within hexlet
X.SHUFFLE.256	Crossbar shuffle within triclet

Fig. 34A

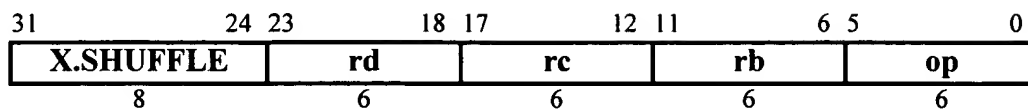
Format

X.SHUFFLE.256 rd=rc,rb,v,w,h

X.SHUFFLE.size rd=rcb,v,w

rd=xshuffle256(rc,rb,v,w,h)

rd=xshufflesize(rcb,v,w)



rc ← rb ← rcb

x ← log₂(size)

y ← log₂(v)

z ← log₂(w)

op ← ((x*x*x-3*x*x-4*x)/6-(z*z-z)/2+x*z+y) + (size=256)*(h*32-56)

Fig. 34B

Definition

```

def CrossbarShuffle(major,rd,rc,rb,op)
  c ← RegRead(rc, 128)
  b ← RegRead(rb, 128)
  if rc=rb then
    case op of
      0..55:
        for x ← 2 to 7; for y ← 0 to x-2; for z ← 1 to x-y-1
          if op = ((x*x*x-3*x*x-4*x)/6-(z*z-z)/2+x*z+y) then
            for i ← 0 to 127
              ai ← c(i6..x || iy+z-1..y || ix-1..y+z || iy-1..0)4..0 of
      0..27:
        cb ← c || b
        x ← 8
        h ← op5
        for y ← 0 to x-2; for z ← 1 to x-y-1
          if op4..0 = ((17*z-z*z)/2-8+y) then
            for i ← h*128 to 127+h*128
              ai-h*128 ← cb(iy+z-1..y || ix-1..y+z || iy-1..0)
            end
          endif
        endfor; endfor
      28..31:
        raise ReservedInstruction
    endcase
  endif
  RegWrite(rd, 128, a)
enddef

```

Fig. 34C

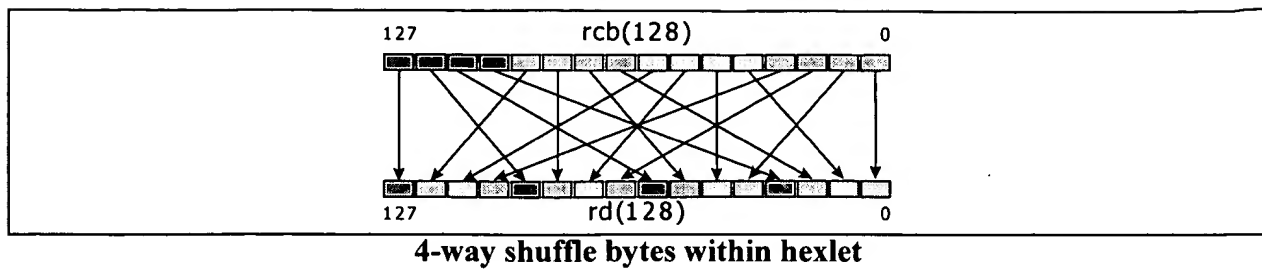


Fig. 34D

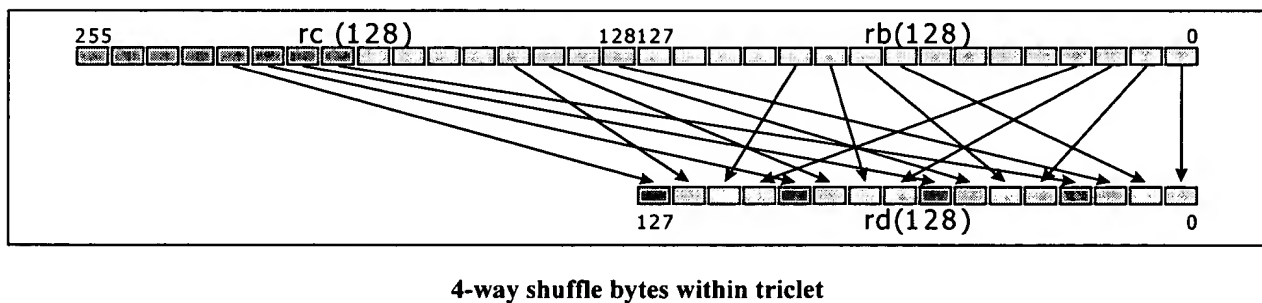


Fig. 34E